

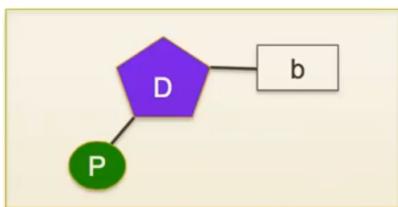
I DATI IN BIOINFORMATICA

- IL GENOMA
 - ACCOPIAMENTI
 - STRINGHE
 - RNA
 - PROTEINE
- IL GENE
 - FORMATO FASTA
- ESPRESSIONE GENICA
 - ALTERNATIVE SPLICING
 - GENE TRANSFER FORMAT
 - CAMPPI FEATURE
 - SPLICING GRAPH
 - ALLINEAMENTO DI SEQUENZE
 - PROGRAMMAZIONE DINAMICA
 - ALLINEAMENTO MULTIPLO
 - PROBLEMA DI OTTIMO
 - MANHATTAN TOURIST PROBLEM
 - LONGEST COMMON SUBSEQUENCE (LCS)
 - HAMMING DISTANCE VS EDIT DISTANCE
 - PSEUDO - CODICE
 - CONFRONTO TRA 3 SEQUENZE
- FILOGENESI
 - FILOGENESI PERFETTA
 - IL PROBLEMA DELLA FILOGENESI PERFETTA
 - LAMINARITÀ
 - RICOSTRUZIONE ALBERO
 - TUMORI
 - APLITIPI
- IL DATO DI SEQUENZIAMENTO
 - IL SEQUENZIAMENTO
 - SHOTGUN SEQUENCING
 - TECNOLOGIE DI SEQUENZIAMENTO - METODO SANGER
 - TECNOLOGIE NGS
 - CONSEGUENZE IMPORTANTI
 - QUALITÀ DEL DATO NGS
 - FASTQ
 - FRAGMENT ASSEMBLY
 - ASSEMBLAGGIO DE NOVO
 - DETERMINAZIONE DEI CONTINGS
 - SHORTEST SUPERSTRING PROBLEM
 - OVERLAP LAYOUT CONSENSUS
 - GRAFO DI DE BRUIJN
 - CONFRONTO
 - SEQUENZIAMENTO PER IBRIDAZIONE
- STRUTTURE DATI IN BIOINFORMATICA
 - HASHING
 - STRUTTURE DI INDICIZZAZIONE
 - NOTAZIONI
 - SUFFIX ARRAY
 - SUFFIX TREE
 - LONGEST COMMON PREFIX ARRAY

I DATI IN BIOINFORMATICA

Iniziamo col DNA (*Acido Deossiribonucleico*) che è una molecola composta da **nucleotidi**

Possiamo vedere un nucleotide composto come in immagine

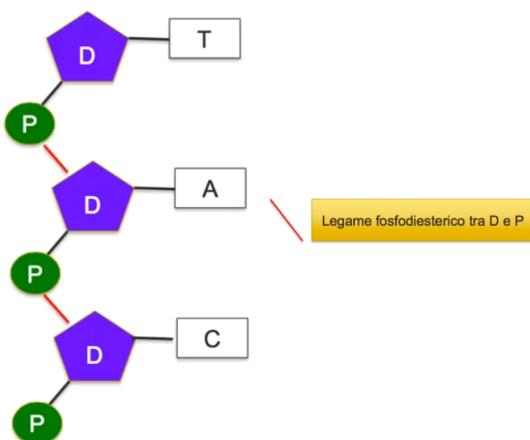


- 3' Con **D** che rappresenta lo zucchero pentoso, nello specifico il *deossiribosio*,
P che rappresenta il gruppo fosfato e
b che rappresenta la base azotata.
- 5' Tutti i nucleotidi strutturalmente sono identici, quello che li differenzia è la base azotata **b** e in particolare ci sono quattro tipi di nucleotidi: **Adenina**, **Citosina**, **Guanina**, **Timina** indicati con $\{A, C, G, T\}$ o con $\{a, c, g, t\}$.

L'Adenina e la Guanina sono le cosiddette **basi purine** mentre la Citosina e la Timina sono dette **basi pirimidine**.

Una direzione basta sui legami biochimici che si instaurano all'interno dei nucleotidi è la cosiddetta **Direzione 5'3'** che va dal gruppo fosfato verso lo zucchero. Questa direzione permette di leggere correttamente la successione delle basi all'interno del DNA.

Vediamo un esempio:



Notiamo che i legamenti tra i vari nucleotidi avvengono attraverso dei **legami fosfodiesterici** tra il gruppo fosfato di un nucleotide e lo zucchero pentoso dell'altro.

La direzione di lettura 5'3' quindi ci dice di leggere dalla direzione dove il gruppo fosfato è in basso e lo zucchero in alto, quindi la sequenza primaria di questa molecola di tre nucleotidi è **C-A-T**.

IL GENOMA

L'unità di misura della lunghezza di una molecola di DNA si chiama *base pair (bp)*.

Il genoma, la molecola di DNA fondamentale per gli organismi, si trova nel nucleo cellulare (*di tutte le cellule*), ha una struttura a doppia elica (*è quindi una doppia catena di DNA che si appaia, da qui il termine base pair*) ed è frammentata in cromosomi. Essa contiene tutte le istruzioni che regolano la vita e lo sviluppo di un organismo.

Ad esempio il *genoma umano* è lungo circa 3.2 miliardi di basi ed è composto da

- 22 autosomi (*chr₁, ..., chr₂₂*)

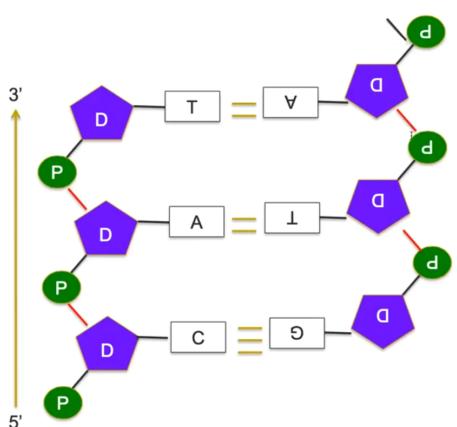
Il cromosoma 1 è il più lungo ed ha circa 245 milioni di nucleotidi e il 22 è il più corto con circa 49 milioni di basi.

- Cromosomi X e Y

In realtà noi siamo degli organismi diploidi e quindi abbiamo 46 cromosomi (*accoppiati*) poiché un corredo viene dal padre e uno dalla madre.

Tuttavia, non c'è nessuna correlazione tra la lunghezza del genoma e la complessità di quest'ultimo e quindi la complessità dell'organismo. Ad esempio mentre una locusta ha un gene lungo circa 5 miliardi di basi, un moscerino della frutta ne ha uno lungo 140 milioni.

ACCOPPIAMENTI



Abbiamo detto che il genoma è costituito da una doppia catena di DNA che si appaia; l'accoppiamento delle basi ha una struttura specifica poiché le basi **A** e **T** generano due legami di idrogeno e quindi si appaiano tra di loro, mentre le basi **C** e **G** ne generano tre. Questa viene chiamata **regola delle basi complementari**.

Quindi se in una catena c'è una **A**, nell'altra ci sarà una **T** e viceversa. Discorso analogo vale per **C** e **G**.

La catena opposta tuttavia ha senso opposto rispetto alla prima, come visibile in foto.

Di solito la prima catena viene chiamata **catena diretta** o *forward strand*, mentre la seconda viene chiamata **catena inversa** o *reverse strand*.

STRINGHE

La sequenza primaria di una catena di DNA è una stringa di simboli sull'alfabeto $\Sigma = \{A, C, G, T\}$ (o $\Sigma = \{a, c, g, t\}$).

Data la sequenza primaria di una delle due catene del DNA genomico, la sequenza primaria della catena appaiata si ottiene con un'operazione chiamata **reverse and complement**.

Ad esempio, data la stringa *ACGTA*

1. Eseguiamo la reverse *ATGCA*
2. Eseguiamo il complemento delle basi *TACGT*

Chiamiamo una sottostringa della sequenza primaria del DNA genomico **sequenza genomica**. Inoltre, una regione di DNA genomico prende il nome di *locus*: la sequenza primaria di un *locus* è una sequenza genomica.

RNA

L'RNA (*acido ribonucleico*) è una molecola composta da nucleotidi proprio come il DNA, difatti ha la stessa struttura solo che come zucchero presenta il ribosio invece del deosiribossio.

Tuttavia, anche le basi dell'RNA cambiano leggermente: esso presenta come basi l'**Adenina**, la **Citina**, la **Guanina** e l'**Uracile**. Tuttavia spesso viene utilizzato comunque l'alfabeto $\Sigma = \{A, C, G, T\}$ per semplicità poiché spesso DNA ed RNA vanno confrontati.

L'RNA inoltre si trova solo in **catene singole**, quindi non sarà mai in doppia catena come il DNA.

PROTEINE

Una proteina è una catena di amminoacidi e la sua sequenza primaria è una stringa su un alfabeto di 20 simboli (*i 20 amminoacidi presenti in natura*).



Vedremo successivamente che il gene, una parte del DNA genomico, opera in un certo modo ed arriva a produrre una proteina.

IL GENE

Il genoma governa la vita di un organismo attraverso la produzione di **proteine** che sono le principali responsabili della struttura e delle attività dell'organismo.

Ogni **proteina** è costruita a partire da un **gene** (*che può però costruire più proteine contrariamente a quanto supposto inizialmente*), che costituisce una piccola parte del genoma.

Generalmente, per ogni milione di basi di genoma, vi sono 11 geni. Occupano quindi una piccola parte del genoma e sono discontinui. Si stima che per gli organismi eucarioti (*come gli umani*) i geni ricoprono dal 2 al 4% del genoma. I geni possono situarsi su entrambe le catene del genoma. Non è detto che se un genoma è lungo abbia tanti geni e i geni stessi sono trasversali rispetto alla specie.

Inoltre, due geni sono omologhi se sono derivati da una differenziazione di specie. Se i due geni si trovano in due specie diverse e quindi derivano da un antenato comune si chiamano geni ortologhi, mentre se abbiamo due geni che derivano da un gene comune ma nella stessa specie si tratta di geni paraloghi.

Un gene quindi è una regione del genoma, chiamata anche *locus*, che esprime una proteina e la sequenza primaria di questo *locus* è la **sequenza genomica** del gene. I geni sono indicati con degli **HUGO NAME**, che sono dei nomi unici dedicati ai geni.

Inoltre, anche i geni sono accoppiati negli organismi diploidi e una coppia di geni viene chiamata *allele*.

FORMATO FASTA

Il formato FASTA è il formato standard utilizzato per le sequenze nucleotidiche e per le proteine. È un formato plain text che memorizza la sequenza primaria e qualche informazione addizionale (*un header su una singola riga*). È nato come formato di input di un software FASTA ed utilizza estensioni `fa` oppure `fasta`.

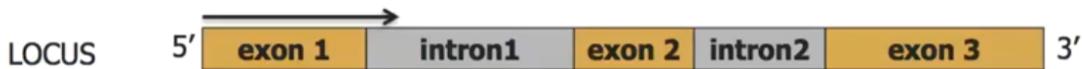
ESPRESSIONE GENICA

Abbiamo detto che un gene è una *regione* di DNA (*locus*) che esprime (*codifica*) una o più proteine, ovvero delle sequenze di aminoacidi.

Questa codifica avviene in due fasi:

1. **Trascrizione + Splicing**: questi primi due step avvengono nel nucleo della cellula e abbiamo un passaggio da locus del gene a **mRNA o RNA messaggero** (*trascritto*).

Vediamo prima come è strutturato il gene:



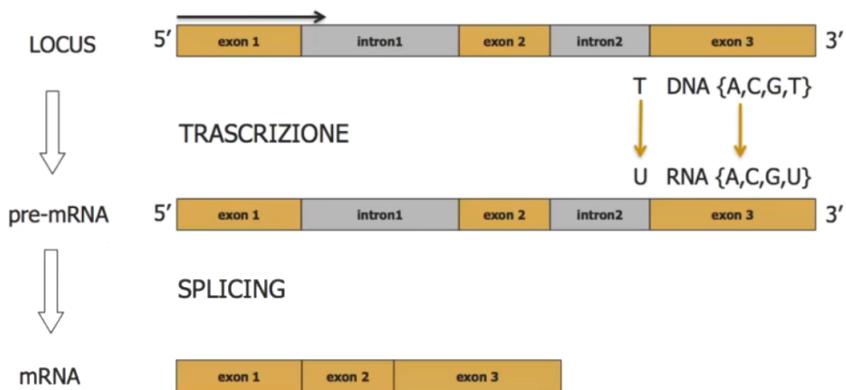
Notiamo che il gene ha una struttura discontinua; è un'alternanza di **regioni codificanti (esoni)** e di **regioni non codificanti (introni)**.

Inoltre, il locus inizia con un esone e finisce con un esone: con n esoni avremmo $n - 1$ introni.

Il confine tra un esone ed un introne successivo viene chiamato **sito di splicing al 5'** o anche *sito del donatore*, mentre il suo duale, ovvero il confine tra un introne ed un esone successivo viene chiamato **sito di splicing al 3'** o anche *sito accettore*.

Il grafico tuttavia è sproporzionato: gli introni, pur avendo una lunghezza variabile (*fino a 20-30 migliaia di basi*), sono generalmente molto più lunghi degli esoni (*lunghi poche centinaia di basi*). Non si sa il motivo esatto, ma sembra che questo aiuti nello splicing.

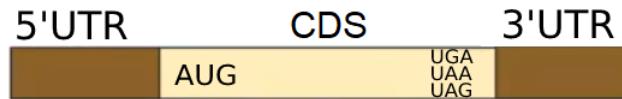
Un'informazione interessante è che, nell'ambito dei geni umani (*anche se non si discosta molto per altre specie di vertebrati*), nel 99.24% dei casi gli introni presentano un determinato pattern: la sequenza primaria dell'introne inizia con **GT** e finisce con **AG**. Quando l'introne rispetta questa regola viene chiamato **introne canonico**.



Nella fase di trascrizione si passa dalla sequenza primaria del locus del gene sul DNA $\Sigma = \{A, C, G, T\}$ a una sequenza primaria su RNA $\Sigma = \{A, C, G, U\}$ che è una copiatura della prima con l'uracile al posto della timina. Chiaramente viene copiata solo la catena interessata visto che l'RNA è un filamento unico. La molecola ottenuta viene chiamata **pre-mRNA**.

Per ottenere poi l'**mRNA** viene effettuato lo **splicing**, cioè vengono eliminati gli introni dal pre-mRNA. Notare che nel grafico i confini degli esoni sono mantenuti per visualizzare la molecola, ma è una molecola singola e continua.

2. **Traduzione:** in questa fase l'mRNA esce dal nucleo e va nel citoplasma dove viene tradotto e si produce la proteina.



Facciamo prima delle considerazioni:

All'interno del mRNA esiste una particolare sottostringa che prende il nome di **Coding Sequence (CDS)** che ha una lunghezza che è un multiplo di tre e quindi è formata da triplette (*chiamati anche codoni*) e che ha delle caratteristiche rigide:

- Inizia con una tripletta **AUG** chiamata anche *codone di inizio*
- Finisce con una tripletta tra **UAG**, **UAA** e **UGA** chiamati anche *codone di stop*

Inoltre, le regioni prima e dopo la CDS prendono il nome di **5'UTR** e **3'UTR**, ovvero regioni non tradotte (*Untranslated Regions*).

Ogni codone mappa uno o più **amminoacidi** (4^3 codoni mappano 20 amminoacidi).

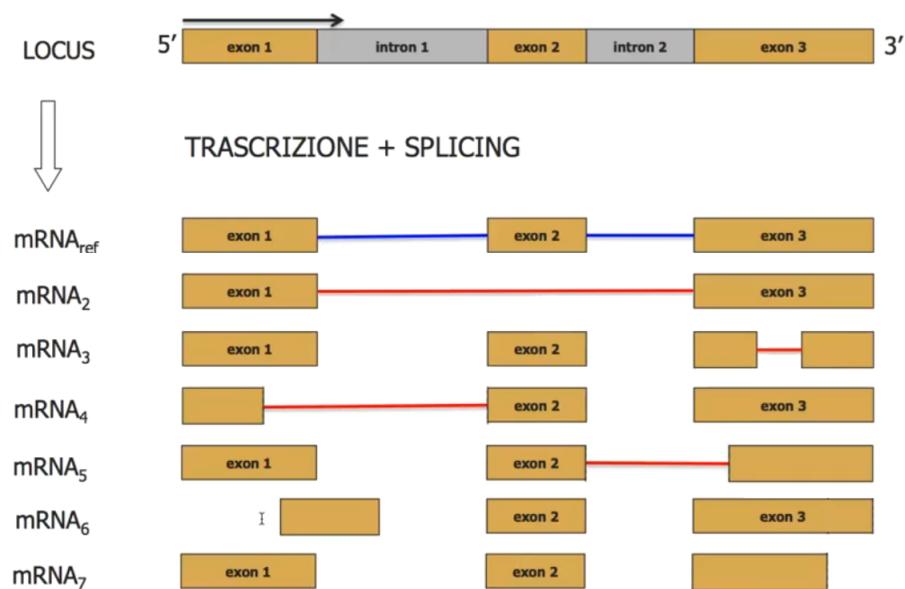
In particolare, il codone **AUG** viene mappato nell'amminoacido **Metionina**; di conseguenza tutte le proteine iniziano con la Metionina. I codoni **UGA**, **UAA** e **UAG** invece si mappano in uno **Stop** che dice quando la scrittura della proteina è terminata, di conseguenza questi codoni non possono trovarsi in mezzo alla CDS. Tuttavia questo non vuol dire che non possiamo trovare le sottostringhe relative, semplicemente se le troviamo esse fanno parte dei codoni adiacenti.

ALTERNATIVE SPLICING

Gli umani hanno circa 20.000 geni protein-coding, mentre le proteine sono centinaia di migliaia. Questa corrispondenza non rispettata è dovuto all'**Alternative Splicing (AS)**, ovvero alla possibilità di un gene di combinare i suoi esoni in modi diversi al fine di esprimere più trascritti (*e quindi più proteine*) mantenendo però l'ordine degli esoni sul locus.

Tuttavia un gene in certe condizioni, in una certa cellula, esprime una sola proteina.

Sono stati tuttavia identificati alcuni pattern:



Prendiamo come riferimento l'mRNA che trascrive tutti gli esoni del locus.

- **EXON SKIPPING:** Osserviamo che un primo pattern, visibile nel mRNA₂, consiste nel evitare di considerare alcuni esoni nella trascrizione dell'mRNA, mantenendo gli altri interi.
- **INTRON RETENTION:** Un altro caso è che un trascritto prenda un prefisso e un suffisso di un esone (*e quindi non considera l'esone nella sua interezza*) come possiamo vedere nell'mRNA₃. Come dice il nome, è come se fosse ritenuto un introne nell'esone spezzato.
Questo pattern può essere visto anche in chiave opposta dove due esoni si fondono generandone uno solo.
- **5' COMPETING SITES:** Questo pattern accade quando viene evidenziato un nuovo sito di splicing al 5'; è come se l'introne si fosse allungato verso sinistra. Possiamo osservarlo nell'mRNA₄.
- **3' COMPETING SITES:** Questo pattern è il duale di quello precedente, dove viene evidenziato un nuovo sito di splicing al 3'. Possiamo osservarlo nell'mRNA₅.
- **MULTIPLE PROMOTERS:** Si verifica quando c'è un *sito promotore*, ovvero quando la trascrizione inizia dopo il sito di splicing al 5'. Possiamo osservarlo nell'mRNA₆.
- **MULTIPLE POLYA:** Evento duale del precedente; il sito di fine si chiama *sito polyA*. Possiamo osservarlo nell'mRNA₇.
- **MUTUAL EXCLUSIVE EXONS:** Esiste un evento più complicato da rilevare che prende il nome di esoni mutualmente esclusivi dove appunto solo uno tra due o più esoni può essere attivo.

GENE TRANSFER FORMAT

Il formato *Gene Transfer Format* o **GTF** è un formato di plain text derivante dal *General Features Format* ed è lo standard che permette di annotare un gene su una *sequenza genomica* di riferimento e fornisce in termini di coordinate:

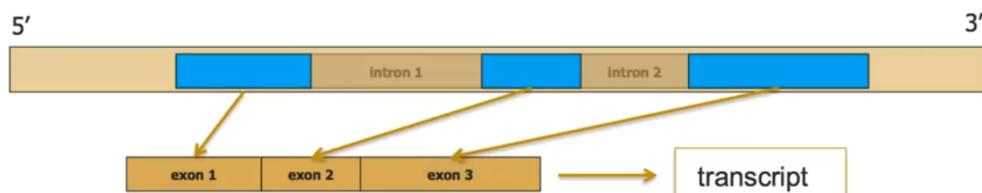
- La composizione in esoni dei suoi trascritti (*tutti*)
- La composizione delle *coding sequences CDS* dei trascritti (*composizione perché il gene è discontinuo sul genoma*)
- La composizione delle 5'UTR dei trascritti
- La composizione delle 3'UTR dei trascritti
- La presenza di start e stop codon delle CDS

Chiaramente la sequenza genomica di riferimento deve essere estratta da una porzione del genoma che copre il *locus* del gene. Tuttavia è possibile annotare anche un gene sulla catena opposta, questo perché generalmente GTF annota più geni da entrambe le catene; ovviamente bisogna considerare le giuste coordinate quando si trascrive un gene dalla catena opposta (*in quel caso le coordinate saranno discendenti percorrendo la genomica di riferimento*).

Questo formato è composto da *record* di nove campi separati dalla tabulazione e la sua estensione è `gtf` o `gff`. Ogni record rappresenta una **feature**, ovvero una regione continua del genoma che può rappresentare:

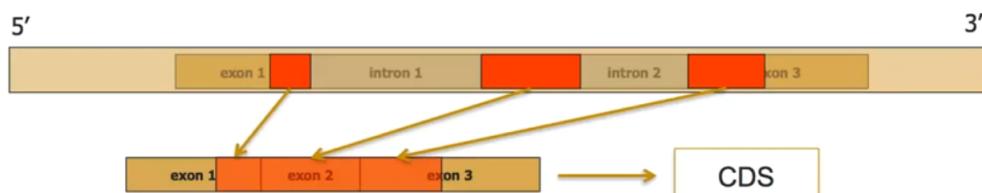
- Un esone con la parola chiave `exon`

In particolare, la feature rappresenta un esone in un solo trascritto: se l'esone appare in un altro trascritto viene descritto in un secondo record.



- Una porzione di CDS con la parola chiave `CDS`

Chiaramente una CDS può espandersi su più esoni in un trascritto, di conseguenza una porzione continua di CDS può richiedere più features (*nell'esempio in foto richiede 3 features*)



- Uno start/stop codon `start_codon` `stop_codon`

Non sono altro che le triplette di inizio/fine di una CDS, di conseguenza si mappano all'inizio della prima feature di una CDS e nella fine dell'ultima.

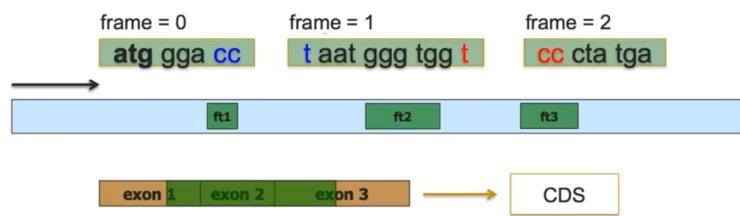
- Una porzione 5'UTR/3'UTR `5UTR` `3UTR`

Potrebbero essere più di una se sono separate da degli introni.

CAMPPI FEATURE

I campi delle feature sono

1. **Identificatore** della genomica di riferimento
2. **Sorgente** che ha prodotto l'annotazione (*ad es. un software*)
3. **Nome feature** (*exon, CDS, start_codon, ...*)
4. **Posizione di inizio** della feature (*1 based*)
5. **Posizione di fine** della feature (*1 based*)
6. **Score** della feature
Ad esempio score di affidabilità se la feature proviene da un software di predizione.
7. **Strand** (+ o -) rappresenta il verso della genomica di riferimento rispetto alla feature annotata.
Per ottenere la sequenza di una feature con strand + bisogna semplicemente estrarre la sottosequenza della genomica di riferimento corrispondente alla feature stessa.
Per ottenere la sequenza di una feature con strand - bisogna estrarre la sottosequenza della genomica di riferimento corrispondente alla feature stessa ed eseguire un'operazione di **reverse & complement** della sottosequenza ottenuta.
8. **Frame** (0, 1, 2) solo per CDS, start_codon e stop_codon
Definita per una feature CDS vale
0 se la prima base della feature è la prima base di un codone
1 se la seconda base della feature è la prima base di un codone
2 se la terza base della feature è la prima base di un codone
Questo campo ci permette di capire come sono separati i codoni nel passaggio da una feature all'altra.
Può essere calcolato come $frame = (3 - L \bmod 3) \bmod 3$
Con L = lunghezza totale delle features a monte di quella calcolata.



9. **Attributi** quelli obbligatori sono *gene_id* che identifica il gene al quale si fa riferimento e il *transcript_id* che identifica il trascritto.
Ha una struttura chiave valore.

SPLICING GRAPH

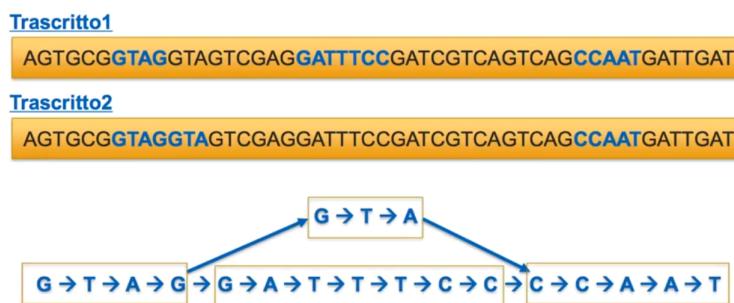
Lo splicing graph è il grafo di rappresentazione dell'insieme dei trascritti espressi da un gene. È un grafo diretto acilcico $G = (V, E)$ dove V è l'insieme delle regioni codificant del gene ed E è la relazione di consecutività (*dei simboli nelle dovute posizioni*) nei trascritti. I trascritti sono percorsi nel grafo ma non vale sempre il viceversa.

Osserviamo un esempio con un solo trascritto:



E un esempio con due trascritti:

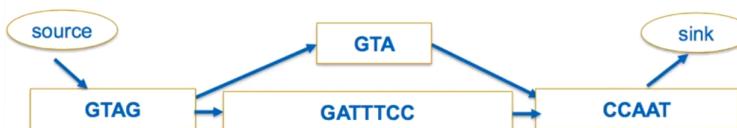
Notare le due G diverse dopo la ramificazione; sono entrambe due G, ma in posizioni diverse



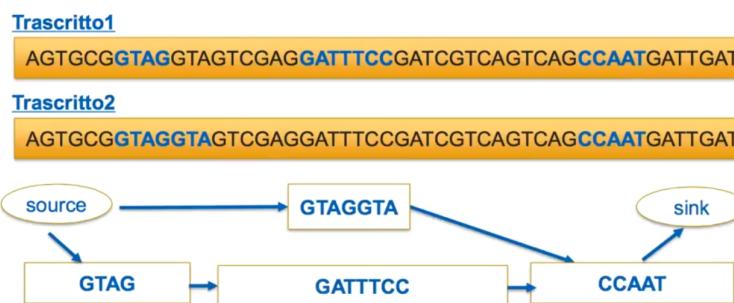
Per semplicità solitamente si collassano i percorsi semplici, cioè quei percorsi con grado entrante uguale a grado uscente ottenendo un grado etichettato dai percorsi semplici piuttosto che da un solo simbolo.

Notiamo che tra i vari nodi, esistono anche nodi che non sono esoni (come GTA).

Spesso si aggiunge poi un nodo *source* dal quale escono le regioni codificant che iniziano un trascritto ed un nodo *sink* nel quale finiscono le regioni che finiscono un trascritto, in modo che tutti i trascritti siano un percorso da *source* a *sink* (*non è vero sempre il viceversa però*).

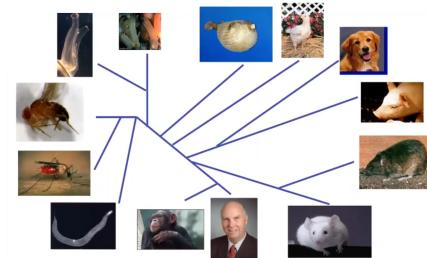


Una formulazione alternativa dello splicing graph è quella che vede V come l'insieme degli **esoni** del gene.



ALLINEAMENTO DI SEQUENZE

L'allineamento di sequenze è la procedura principe con cui i biologi analizzano le sequenze biologiche. Una delle applicazioni di questa procedura è ad esempio la creazione di alberi filogenetici che non hanno una radice poiché rappresentano solo i legami tra le specie attuali rappresentate dalle foglie (*si ipotizza che gli antenati sono specie ormai scomparse*). Un'altra utilità dell'allineamento è la possibilità di mettere a confronto sequenze di DNA, RNA o proteine per scoprirne elementi comuni per individuare mutazioni o varianti nelle specie virali piuttosto che batteriche.



Un **paradigma** di questo processo è che la somiglianza a livello di sequenze (*detta anche similarità*) implica la *stessa funzione* da parte delle sequenze. Su questo paradigma si basano gli esperimenti sugli animali da laboratorio ad esempio.

Questo è però un paradigma direzionale: la stessa funzione di due sequenze non implica necessariamente che siano identiche.

Confrontare le sequenze significa andare ad analizzare il processo di evoluzione che avviene al livello del DNA. Confrontare le sequenze significa quindi risalire ai processi evolutivi avvenuti che sono principalmente le **mutazioni**.

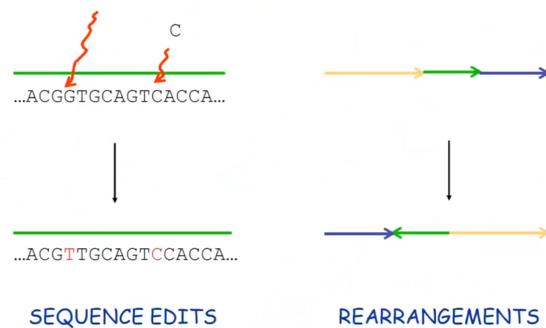
Esistono due tipi di mutazioni:

1. *Mutazioni a livello nucleotidico* che riguardano le singole basi.

Queste sono **operazioni di edit** (*sostituzione di una lettera, inserimento o cancellazione*)

2. *Mutazioni a livello di cromosomi* che riarrangiano il DNA.

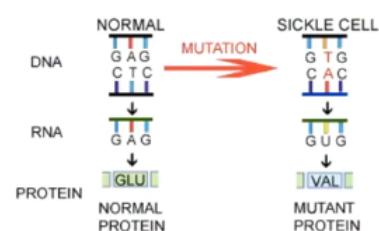
Esistono dei meccanismi all'interno del DNA che lo fa avvolgere attorno a sé stesso e nell'avvolgersi alcuni legami chimici vengono interrotti portando al cambio di orientazione di alcune parti di esso.



Chiaramente le mutazioni a livello di DNA portano poi alla produzione di proteine diverse che determinano poi l'evoluzione della specie stessa. Successivamente, in base alla selezione naturale, solo alcune mutazioni sopravvivono.

Anche una singola mutazione può avere effetti giganteschi; ad esempio una mutazione nel gene *HBB* porta alla produzione di una proteina diversa che determina l'anemia mediterranea che può anche essere fatale.

Va osservato tuttavia che la stessa patologia può essere associata a variazioni diverse, quindi non è detto, ad esempio, che tutte le forme di anemia siano date da questa specifica mutazione ed è questo il motivo per il quale va fatta un'indagine genetica per poter personalizzare la medicina in base alla mutazione.



Vediamo un esempio di allineamento di sequenze:

AGGCTATCACCTGACCTCCAGGCCGATGCC	A	T	-	G	T	T	A	T	-
TAGCTATCACGACC CGCGT GATTGCCCGAC	A	T	C	G	T	--	A	--	C
-AGGCTATCACCTGACCTCCAGGCCGATGCC									
-TAG-CTATCAC--GACC GCG-									
GGTCGATTGCCCGAC									

4 matches 2 deletions
2 insertions

L'allineamento può essere visto o come massimizzazione di coppie uguali o come minimizzazione di coppie in missmatch. Sostanzialmente quello che viene fatto è prendere le due sequenze e inserirne degli spazi (*gap*) al fine di evidenziare le parti di match e le parti di missmatch. Solitamente questa operazione viene eseguita tramite delle matrici.

Chiaramente possiamo vedere un missmatch come un'inserzione in una delle due stringhe o come una delezione nell'altra (*sono operazioni simmetriche*).

Alle mutazioni viene assegnato uno schema di punteggio stabilito attraverso lo studio dell'evoluzione delle probabilità che una base diventi un'altra.

PROGRAMMAZIONE DINAMICA

La costruzione delle matrici usate per l'allineamento delle sequenze avviene per programmazione dinamica.

Innanzitutto l'allineamento di due sequenze S_1 e S_2 su Σ^* può essere definito come una matrice A di dimensione $2 \times l$ ($l \leq m + n$ con $|S_1| = n$ e $|S_2| = m$) in cui

- La prima riga contiene i caratteri di S_1 intervallati da $-$ (*gap*)
- La seconda riga contiene i caratteri di S_2 intervallati da $-$

L'unico vincolo che ha questa matrice è che non deve avere colonne di soli *gap*.

Definiamo la funzione $\delta : (\Sigma \cup \{-\})^2 \rightarrow \mathbb{R}^+$ che assegna ad ogni coppia di simboli un valore, ovvero il **punteggio** citato prima (*esistono anche versioni con punteggi negativi*). Questa funzione quindi definisce una matrice che assegna ad ogni coppia un certo *score*; se vogliamo minimizzare le differenze assegnerà 0 ai simboli uguali e valori diversi da zero negli altri casi (*ad esempio 1*).

Il problema dell'allineamento ottimo è quindi definito come:

INPUT: S_1, S_2, δ

OUTPUT: matrice A di allineamento per S_1 ed S_2 tale che il costo di A , indicato con $c(A)$, sia minimo (*o massimo*)

Dove $c(A)$ è l'applicazione di δ a tutte le colonne della matrice A , ovvero $c(A) = \sum_{i=1}^l \delta(a_{1i}, a_{2i})$

NOTA

Notiamo che possiamo ottenere la lunghezza della *LCS* (*Longest Common Subsequence*) di due stringhe conoscendo la loro matrice di allineamento A ; se impostiamo il problema di allineamento come un problema di massimo (*e quindi evidenziamo le coppie uguali, assegnando peso unitario alle coppie uguali e peso 0 alle altre*), la lunghezza della *LCS* sarà pari al costo di A . Possiamo poi ottenere la *LCS* andando a prendere solo i simboli tali per cui δ vale 1.

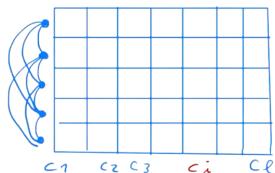
ALLINEAMENTO MULTIPLO

Il problema si può estendere a più sequenze e prende il nome di **allineamento multiplo**, creando una matrice $k \times l$.

Tuttavia, δ rimane sempre la stessa e quindi bisogna definire $c(A)$; il costo della matrice A è definito sempre come la somma dei costi delle sue colonne $c_1 \dots c_l$.

Esistono diverse definizioni di $c(A)$ che dipendono da diversi obiettivi:

- **SUM OF PAIRS**



Dobbiamo definire cos'è il costo della colonna c_i e per farlo assegnamo ad ogni cella della colonna un nodo di un grafo e, per ogni coppia di nodi v_1 e v_2 , tracciamo un arco (*creando un grafo completo*) con peso $\delta(v_1, v_2)$.

In questa definizione il costo di A è il costo del grafo completo. Ovvero $c(A)$ è la somma di tutte le coppie della colonna.

NOTA: sotto questo criterio (*con algoritmo di minimizzazione*) una colonna di simboli uguali ha peso 0, mentre una colonna con un solo simbolo diverso dagli altri avrà costo $k - 1$ perché, essendo completo, il nodo diverso è collegato a tutti gli altri $k - 1$ nodi.

- **CONSENSUS (O STAR ALIGNMENT)**



L'idea del consensus è scegliere una sequenza di riferimento S^* e confronto tutte contro di essa.

In questo caso il grafo diventa un albero con radice il simbolo della colonna corrispondente di S^* . Il peso è sempre il costo del grafo.

- **TREE - ALIGNMENT**

Un'altra definizione non così tanto utilizzata come le precedenti due è la tree - alignment che basa il suo confronto sulla base di un albero di evoluzione costruito tra le varie sequenze S .

PROBLEMA DI OTTIMO

Possiamo ora definire il problema del calcolo della matrice A tale che il suo costo $c(A)$ sia ottimo.

La soluzione a questo problema è una basata sulla programmazione dinamica è

- Per 2 sequenze, ha un tempo $O(n \cdot m)$ con $|S_1| = n$ e $|S_2| = m$, ovvero $O(l^2)$ se $|S_1| = l = |S_2|$

Dato il costo comunque elevato, spesso si considerano anche metodi **alignment - free** che confrontano le sequenze senza farne l'allineamento.

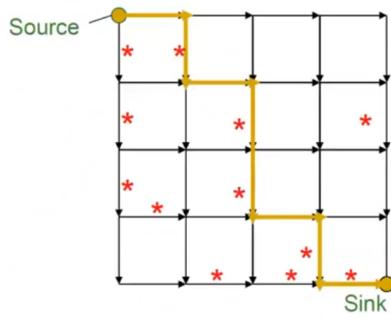
- Per k sequenze dipende dal criterio di calcolo del costo delle colonne e generalmente è alto ed è dell'ordine di $O(l^k)$

Tuttavia, senza conoscere il numero di sequenze in input il problema è addirittura NP - Hard (*anche quando prendiamo uno scoring δ banale che soddisfa la disuguaglianza triangolare e anche per un alfabeto binario*).

Essendo un problema difficile si sono quindi sviluppate delle **euristiche**.

Al giorno d'oggi si sta passando sempre più spesso da metodi basati sulla programmazione dinamica a metodi basati sulla *BWT*.

MANHATTAN TOURIST PROBLEM

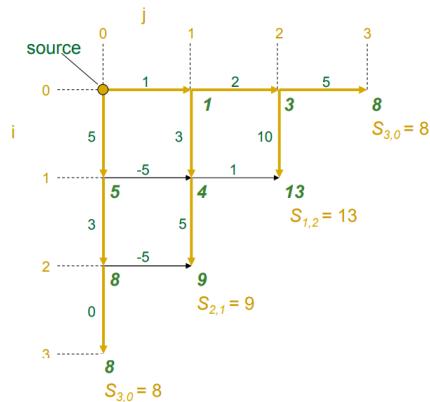


Il problema dell'allineamento si può ridurre al problema di trovare un cammino in una griglia, che è conosciuto anche come Manhattan Tourist Problem (*per via della struttura geografica a griglia delle strade di Manhattan*) che consiste nel trovare un percorso dalla sorgente al pozzo che massimizzi il numero di attrazioni turistiche (*indicate con **) visitate durante il percorso andando solo a sud o a est. Possiamo vedere le attrazioni turistiche come un peso, in modo da cercare un cammino di peso massimo.

Quindi il problema MTP è formulato come segue:

INPUT: Una griglia pesata G con due vertici distinti *sorgente* e *pozzo*

OUTPUT: Il cammino di peso massimo in G dalla *sorgente* al *pozzo*

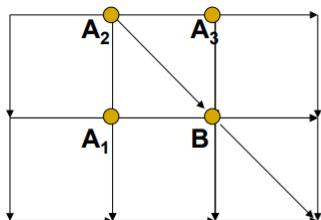


Chiaramente per questo problema un approccio *greedy* non è ottimale in quanto scegliere subito la strada localmente migliore potrebbe precludere strade migliori, per questo motivo si utilizza la programmazione dinamica che adotta un approccio bottom - up che parte dalla sorgente e analizza le possibili strade e si segna man mano che avanza il massimo raggiungibile in ogni nodo.

In particolare l'algoritmo si segna per ogni nodo

$$s_{ij} = \max \begin{cases} s_{i-1,j} + \text{weight of edge } < (i-1, j), (i, j) > \\ s_{i, j-1} + \text{weight of edge } < (i, j-1), (i, j) > \end{cases}$$

A volte la griglia può presentare anche percorsi diagonali da un nodo all'altro.



$$s_B = \max \text{ of } \left\{ \begin{array}{l} s_{A1} + \text{weight of the edge } (A_1, B) \\ s_{A2} + \text{weight of the edge } (A_2, B) \\ s_{A3} + \text{weight of the edge } (A_3, B) \end{array} \right\}$$

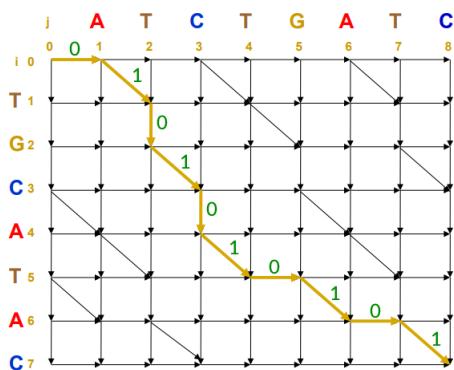
LONGEST COMMON SUBSEQUENCE (LCS)

Date due sequenze $v = v_1 \dots v_m$ e $w = w_1 \dots w_n$ la **LCS** di v e w è una sequenza di posizioni in v $1 \leq i \leq m$ e una sequenza **massimale** di posizioni in w $1 \leq j \leq n$ tali che la i -esima lettera di v è identica alla j -esima lettera in w .

<i>i</i> coords:	0	1	2	2	3	3	4	5	6	7	8
elements of v	A	T	-	C	-	T	G	A	T	C	
elements of w	-	T	G	C	A	T	-	A	-	C	
<i>j</i> coords:	0	0	1	2	3	4	5	5	6	6	7

In particolare in questo esempio le posizioni in v sono 2 3 4 6 8 e le posizioni in w sono 1 3 5 6 7.

Come abbiamo detto anche in precedenza, una matrice di allineamento evidenzia la sottosequenza più lunga.



Possiamo risolvere il problema della *LCS* con la programmazione dinamica cercando un grafo pesato come quello del *MTP* con anche le diagonali e trovando il cammino massimo dalla sorgente al nodo.

Formalmente:

INPUT: Un grafo pesato G con una *sorgente* e un *pozzo*

OUTPUT: Il cammino di peso massimo dalla *sorgente* al *pozzo*

Il peso sarà 1 per le diagonali e 0 per gli altri archi.

Notiamo in particolare il significato attribuito alle frecce: le diagonali sono usate per *matchare* due simboli, la freccia in basso significa prendere un simbolo della sequenza w e nessuno della v e viceversa per la freccia orizzontale; quello che vogliamo ottenere è un percorso col maggior numero di diagonali. Questo ci permette di creare una matrice di allineamento man mano che attraversiamo il percorso.

La lunghezza della *LCS* è calcolata come

$$s_{i,j} = \begin{cases} s_{i-1, j} \\ s_{i, j-1} \\ s_{i-1, j-1} + 1 & \text{if } v_i = w_j \end{cases}$$

Chiaramente, se vogliamo avere un numero di *gap* limitato possiamo anche costruire solo una parte della matrice intorno alla diagonale (*banda*).

HAMMING DISTANCE VS EDIT DISTANCE

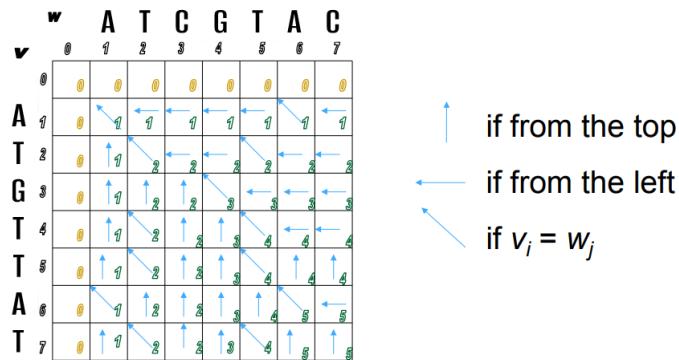
Osserviamo le due sequenze di DNA v e w e notiamo che hanno una distanza di Hamming, definita come il numero di posizioni con simboli diversi, pari a $d_H(v, w) = 8$, tuttavia se shiftiamo una delle due sequenze di una posizione otteniamo $d(v, w) = 2$, dove d è la *distanza di edit* definita come il minimo numero di operazioni elementari (*inserzioni*, *delezioni* e *sostituzioni*) necessarie per trasformare una stringa in un'altra.

$$\begin{array}{ll} v : \textcolor{red}{A} \textcolor{blue}{T} \textcolor{red}{A} \textcolor{blue}{T} \textcolor{red}{A} \textcolor{blue}{T} & v : \textcolor{red}{A} \textcolor{blue}{T} \textcolor{red}{A} \textcolor{blue}{T} \textcolor{red}{A} \textcolor{blue}{T} \textcolor{red}{A} \textcolor{blue}{T} \\ w : \textcolor{red}{T} \textcolor{blue}{A} \textcolor{red}{T} \textcolor{blue}{A} \textcolor{red}{T} \textcolor{blue}{A} & w : \textcolor{red}{-} \textcolor{blue}{T} \textcolor{red}{A} \textcolor{blue}{T} \textcolor{red}{A} \textcolor{blue}{T} \textcolor{red}{A} \end{array}$$

La distanza di edit è un caso particolare dell'allineamento multiplo, quindi si può calcolare la distanza di edit tramite il calcolo dinamico della *LCS*.

PSEUDO - CODICE

Notiamo inanzitutto che possiamo salvare, per ogni cella della matrice, oltre al valore massimo ottenibile, il punto di origine di questo massimo in una seconda matrice B ; questo sarà utile per ricostruire poi la LCS in tempo lineare ripercorrendo le frecce dal nodo *pozzo*.



Per riempire la matrice ci vuole un tempo $O(n \cdot m)$ con la programmazione dinamica (*notare i due for annidati*).

```

LCS(v, w)
    for i = 1 to n
        s[i][0] = 0
    for j = 1 to m
        s[0][j] = 0

    for i = 1 to n
        for j = 1 to m
            if v[i] == w[j]
                s[i][j] = max(s[i-1][j], s[i][j-1])
            else
                s[i][j] = max(s[i-1][j], s[i][j-1], s[i-1][j-1] + 1)

            if s[i][j] == s[i-1][j]      B[i][j] = ↑
            if s[i][j] == s[i][j-1]      B[i][j] = ←
            if s[i][j] == s[i-1][j-1]   B[i][j] = ↖

    return (s, B)

```

Per visualizzare poi la LCS scorriamo la matrice B

```

PrintLCS(B, v, i, j)
    if i == 0 || j == 0
        return
    if B[i][j] == ↖
        PrintLCS(B, v, i - 1, j - 1)
    if B[i][j] == ←
        PrintLCS(B, v, i, j - 1)
    if B[i][j] == ↑
        PrintLCS(B, v, i - 1, j)

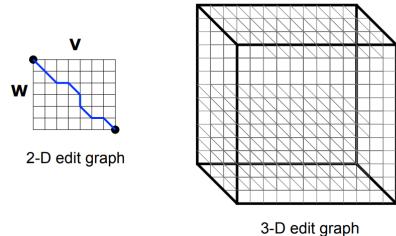
    print(v[i])
    return

```

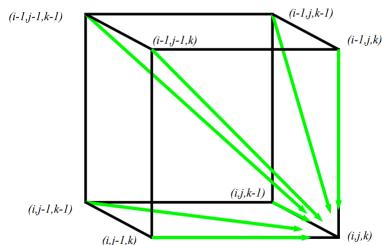
CONFRONTO TRA 3 SEQUENZE

Vediamo ora come generalizzare il problema a più di due sequenze e, nello specifico, per tre sequenze; l'idea è adattare il *MTP* ad un caso tridimensionale immaginando di potersi spostare su tre dimensioni.

Dal punto di vista biologico viene usato l'allineamento multiplo per via dell'evoluzione e, a causa di essa, succede spesso che la somiglianza debole tra due sequenze diventi molto forte se vengono allineate più sequenze; gli elementi multipli rivelano somiglianze subdole che l'allineamento a coppie non sempre rileva.



Abbiamo visto che usiamo una matrice a due righe per rappresentare l'allineamento di due sequenze e, allo stesso modo, per allinearne tre usiamo una matrice a tre righe; di conseguenza il cammino che troveremo con l'*MTP* avrà tre coordinate.



Chiaramente, aumentando di dimensione, abbiamo anche più cammini che portano ad un punto: se nel caso 2D avevamo 3 lati possibili per ogni punto, nel caso 3D ne abbiamo 7 (*da sopra, da sinistra, dalla diagonale, da dietro e dalle diagonali delle facce che toccano il punto*).

Di conseguenza, il calcolo di ogni nodo diventa $s_{i,j,k} = \max$

$$\left\{ \begin{array}{ll} s_{i-1,j-1,k-1} & +\delta(v_i, w_j, u_k) \\ s_{i-1,j-1,k} & +\delta(v_i, w_j, _) \\ s_{i-1,j,k-1} & +\delta(v_i, _, u_k) \\ s_{i,j-1,k-1} & +\delta(_, w_j, u_k) \\ s_{i-1,j,k} & +\delta(v_i, _, _) \\ s_{i,j-1,k} & +\delta(_, w_j, _) \\ s_{i,j,k-1} & +\delta(_, _, u_k) \end{array} \right.$$

Notiamo che abbiamo esplicitato il peso calcolato dalla funzione δ . Nel caso 2D non lo avevamo esplicitato poiché, assegnando solo alle diagonali peso unitario, $\delta(x, _)$ era pari a 0.

Per tre sequenze di lunghezza n l'algoritmo dinamico ha un tempo di circa $7n^3$, ovvero $O(n^3)$. Per k sequenze si costruisce un *MTP* k -dimensionale con un tempo $(2^k - 1)(n^k)$, ovvero $O(2^k n^k)$.

CONCLUSIONE: l'algoritmo di programmazione dinamica per due sequenze è estendibile facilmente a k sequenze, ma risulta impraticabile per via del tempo esponenziale.

Chiaramente un allineamento multiplo induce l'allineamento a coppie delle varie componenti: non è banale però passare dagli allineamenti a coppie a quello multiplo e non sempre è possibile. Solitamente per l'allineamento multiplo vengono usate delle heuristiche che non dipendono dalla programmazione dinamica.

FILOGENESI

Con *filogenesi* ci riferiamo agli alberi evolutivi, ovvero degli alberi che descrivono le sequenze di eventi che hanno portato alle specie attuali. Questi alberi sono creati appunto dall'**evoluzione** che è dettata dalla *diversità degli individui* e dalle *mutazioni*, ovvero dai cambiamenti nelle sequenze di DNA.

I dati utilizzati per creare questi alberi sono di diverso tipo; abbiamo sia dati **morfologici** (*osservabili come numero di gambe, lunghezza, ecc*) che dati **molecolari** come le *sequenze geniche* e le *sequenze proteiche*. Generalmente la radice di questi alberi (*se creata*) è rappresentata da un'insieme di specie distanti dalle specie di interesse.

Le assunzioni usate per creare questi alberi stabiliscono che gli organismi vicini hanno genomi simili, che i geni simili hanno lo stesso antenato, che esiste un antenato universale per tutte le specie (*tuttavia questa cosa non è provata*) e che le differenze molecolari in geni con lo stesso antenato sono correlate al tempo di evoluzione.

A livello teorico esiste un **evento di speciazione** che consiste nella creazione di specie differenti.

Le sequenze di geni o di proteine possono essere omologhe (*con antenati comuni*) per ragioni differenti:

- A causa di un evento di *speciazione*: questi geni prendono il nome di **geni ortologhi**
- A causa di un evento di *duplicazione*: **geni paraloghi**
L'evento di duplicazione si verifica a causa dell'influenza dell'ambiente e porta alla duplicazione della parte del DNA che descrive il gene in altre parti del genoma. Questi eventi tuttavia non creano nuove specie.
- A causa di un evento di *trasferimento orizzontale* (*e.g. scambio di materiale genetico tra virus e ospite*): **geni xenologhi**

I principali metodi di ricostruzione filogenetica sono

- **Distanza**: combina ricorsivamente due nodi a minima distanza.
Per la distanza vengono usate delle matrici e si usano dei metodi di clustering per a combinazione dei nodi. Le matrici che descrivono le distanze soddisfano la disuguaglianza triangolare $d(i, k) \leq d(i, j) + d(j, k)$ e descrivono la distanza come una metrica, ovvero $d(i, i) = 0$, $d(i, j) > 0$ per $i \neq j$ e $d(i, j) = d(j, i)$.
- **Parsimonia**: si basano sulla riduzione al minimo del numero di cambiamenti.
- **Maximum Likelihood**: costruisce l'albero più probabile rispetto ad un certo modello.

Abbiamo visto come un obiettivo del mondo della bioinformatica sia quello di ricostruire l'**albero della vita**, ovvero l'albero di tutte le specie. Per fare ciò è necessaria una **riconciliazione** dell'evoluzione, ovvero è necessario trovare un *agreement* tra alberi diversi per le specie; questo non è necessario solo per via dei diversi alberi per le varie specie, ma anche perché metodi (*algoritmi*) diversi producono alberi diversi.

Un aspetto fondamentale è che l'evoluzione non riguarda solo le specie; si usano gli alberi anche per inferire **aplotipi** da **genotipi**, per studiare l'**evoluzione dei tumori** (*usano degli alberi i cui nodi interni sono etichettati anche loro*) e anche per analizzare più sequenze.

FILOGENESI PERFETTA

Definiamo il **principio della parsimonia** che definisce i metodi utilizzati per la filogenesi perfetta. I **metodi parsimonia** assumono che ogni specie sia caratterizzata da **caratteri** o **attributi di stati**. Si costruisce un **albero di massima parsimonia** tale che abbia come foglie gli stati dei caratteri associati alle specie in input, mentre i nodi interni vengono etichettati con i caratteri inferiti (*ovvero delle specie antenate*) con i relativi cambiamenti lungo i rami e questi cambiamenti sono minimizzati.

Ma cos'è quindi un **carattere**?

Definiamo come **fenotipo** un carattere visibile (*ad esempio la presenza di gambe, di ali, ecc*).

Esistono tuttavia anche caratteri **genomici**, ovvero informazioni molecolari come ad esempio gli **Single Nucleotide Polymorphism (SNP)** che rappresentano le posizioni nel DNA nelle quali la popolazione differisce per lo 0.01%.

I metodi cercati sono del tipo **Whole Genome Scale (WGS)** poiché prendendo una scala intera genomica possiamo avere vari caratteri genomici come

- Combinazioni di proteine, ordinamento di un gene, composizione di un gene
- Cambiamenti genomici rari
- Inserzioni e delezioni di introni
- Single Nucleotide Polymorphism (SNP) che caratterizzano gli aplotipi rispetto ai genotipi

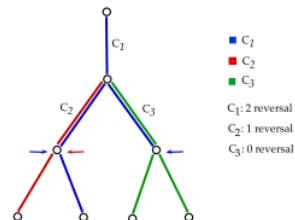
Ovviamente la scelta dei caratteri determina la correttezza dell'albero ottenuto (*ad esempio se vogliamo un albero di evoluzione degli aplotipi andiamo ad usare gli SNP*).

I caratteri genomici sono **binari** e quindi hanno due stati: 0 (*assenza del carattere*) o 1 (*presenza del carattere*).

Esistono vari principi di parsimonia come:

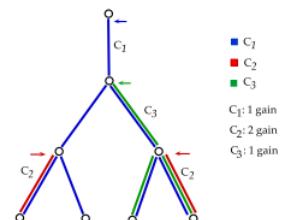
(NOTA: linput del metodo sono i nodi foglia)

- **DOLLO PARSIMONY**: in questo principio un carattere *c* può cambiare dallo stato 0 allo stato 1 una sola volta, ma può cambiare più volte dallo stato 1 allo stato 0 (*reversal*).
(Osserviamo c₁)



- **CARMIN - SOKAL PARSIMONY**: in questo principio un carattere può cambiare dallo stato 0 allo stato 1 più volte, ma non può fare reversal. *(Osserviamo c₂)*

Solitamente questo principio viene evitato per le filogenesi tumorali poiché si assume che in quel caso non vi siano mutazioni su rami differenti (*la si sta mettendo in dubbio ultimamente*).



- **FILOGENESI PERFETTA**

La combinazione dei principi visti prima porta a quella che è chiamata filogenesi perfetta, ovvero un principio in cui una carattere può cambiare da 0 a 1 una sola volta e non può mai avere reversal. È interessante notare che risolvere la filogenesi perfetta richiede tempo lineare, mentre risolvere gli altri due principi è un problema NP - Completo.

IL PROBLEMA DELLA FILOGENESI PERFETTA

Abbiamo quindi detto che l'albero di una filogenesi perfetta per una matrice binaria M di n specie con m caratteri è tale che

- Ogni nodo x dell'albero è annotato con un vettore l_x lungo m dove $l_x[j]$ è lo stato del carattere c_j
- La radice è annotata da un vettore 0 lungo m
- Per ogni carattere c_j c'è al più (*esattamente, non ha molto senso codificare una caratteristica che non appare mai*) un lato e , etichettato con c_j , dove c_j cambia stato da 0 a 1
- Ogni riga della matrice M rappresenta un nodo foglia dell'albero

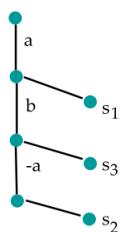
	C_1	C_2	C_3	C_4	C_5
s_1	1	1	0	0	0
s_2	0	0	1	0	0
s_3	1	1	0	0	1
s_4	0	0	1	1	0

Il problema della filogenesi perfetta è definito come segue

INPUT: Una matrice binaria $n \times m$ dove n sono le specie sulle righe s_i ed m i caratteri sulle colonne c_i

OUTPUT: Un albero di filogenesi perfetta, *se esiste* definito come descritto prima.

Chiaramente non sempre esiste questo albero. Pensiamo a due specie che presentano rispettivamente il carattere $c_1 = 1$ $c_2 = 0$ e $c_1 = 0$ $c_2 = 1$; chiaramente non possiamo avere una terza specie che presenti entrambi i caratteri $c_1 = 1$ $c_2 = 1$, poiché questo implicherebbe il passaggio di stato di uno dei due caratteri da 0 a 1 più di un'unica volta.



Quindi la più piccola **matrice proibita** per la filogenesi perfetta è $\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$

Un modello più rilassato che ammette la matrice proibita è quello dei **caratteri persistenti** che permette ad un carattere di fare una *back mutation*, ovvero permette ai caratteri di cambiare da 0 a 1 una sola volta e da 1 a 0 una sola volta.

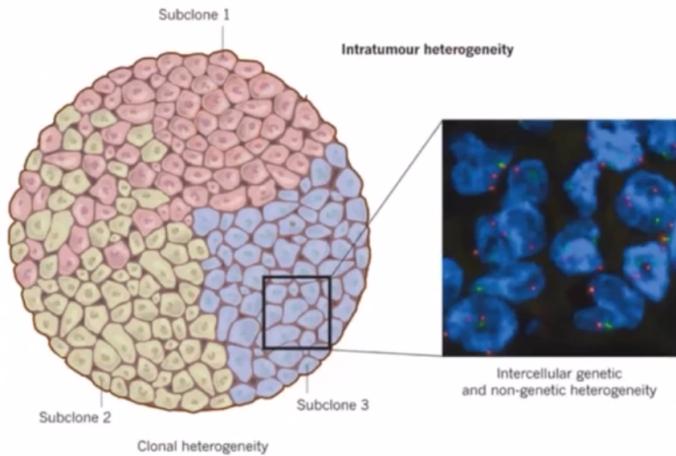
Questo modello è chiamato anche **Dollo - 1** perché permette appunto una *back mutation*. La filogenesi perfetta è una **Dollo - 0**.

INFO: Al momento non è chiaro se il problema Dollo - 1 sia NP - Completo o meno, in Bicocca si sta provando a dimostrare che in realtà è in P. Questo problema viene usato attualmente nelle filogenesi tumorali.

La Dollo - K con $k \geq 2$ è NP - Completo, ma esistono degli algoritmi parametrici.

TUMORI

Un tumore è una conseguenza dell'accumulo di mutazioni (*a seguito di una mutazione driver*) che creano delle popolazioni cellulari accomunate dallo stesso set di mutazioni. Le caratteristiche dei tumori è che sono composti da sotto - cloni diversi.



Generalmente per individuare le mutazioni si confronta un allineamento di alcune read con un reference e si analizza il numero di volte in cui si presenta un cambiamento.

Si costruisce poi una matrice **Variant Allele Frequency**

Frequency rappresentante i caratteri $c_1 \dots c_n$ (*che sono le mutazioni*) e le loro frequenze rispetto a dei campioni, ovvero a delle collezioni di read.

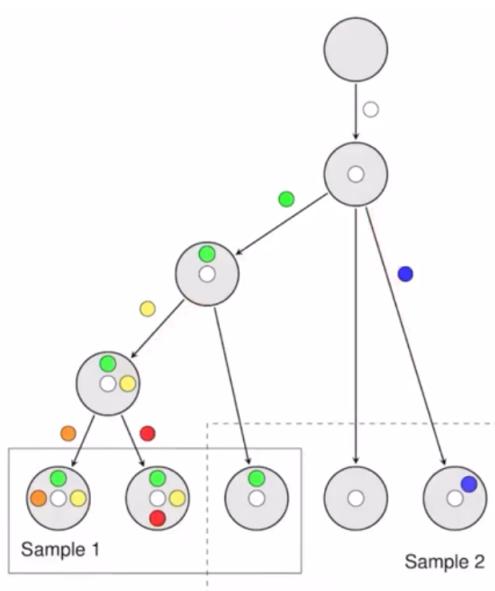
VAF	c_1	c_2	...	c_m
MethX	●	●		●
sample 1	0.6	0.3	0.0	0
sample 2				
sample 3				

Dalla matrice delle frequenze si cerca poi di ricostruire la composizione del campione e dei cloni che contiene e stabilirne una gerarchia, ovvero un ordine in termini evolutivi e quindi costruire l'albero di evoluzione delle mutazioni che si ottiene dalla scomposizione della matrice delle frequenze nelle matrici B e U .

Notiamo che i cloni possono apparire anche nei nodi interni dell'albero; osservando i cloni alle foglie è come se osservassimo l'evoluzione dei cloni.

Matrix B (tree T)						
●	●	○	○	●	●	●
0	0	1	0	0	0	1
0	1	1	1	1	1	0
0	1	1	0	0	0	0
0	0	1	0	0	0	0
1	1	1	1	0	0	0

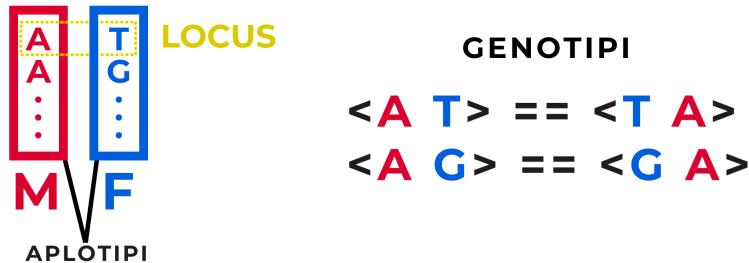
Usage matrix U					
	Clones				
0	0.2	0.2	0	0.2	0
0.4	0	0.4	0.2	0	0



APLOTIPI

Per introdurre gli aploidi dobbiamo prima parlare dei cromosomi: gli umani, che sono diploidi, hanno 23 coppie di cromosomi che sono quindi composti da due set, uno derivante dal padre e uno dalla madre. Il genoma umano è la collezione dei cromosomi e ogni coppia è costituita da un aplotipo materno e da un aplotipo paterno.

Un **aplotipo** di un cromosoma è una sequenza di *nucleotidi* e ogni posizione prende il nome di *locus* il cui valore è un allele specifico. Il **genotipo** è la coppia di valori per locus di due aploidi (senza un ordinamento tra loro).



Le posizioni dove la coppia è costituita da alleli identici si dicono **omozigote**, altrimenti si dicono **eterozigote**. Di particolare interesse negli aploidi sono gli **Single Nucleotide Polymorphism (SNPs)**, ovvero i nucleotidi che caratterizzano la diversità nella popolazione. In particolare, si ha uno SNP quando nel 99.9% dei casi tutti gli individui hanno una certa base in una data posizione, avendo il cosiddetto **allele di maggioranza**, mentre lo 0.1% ne ha una diversa, avendo l'**allele di minoranza** (si rilevano confrontando una popolazione).

ESEMPIO

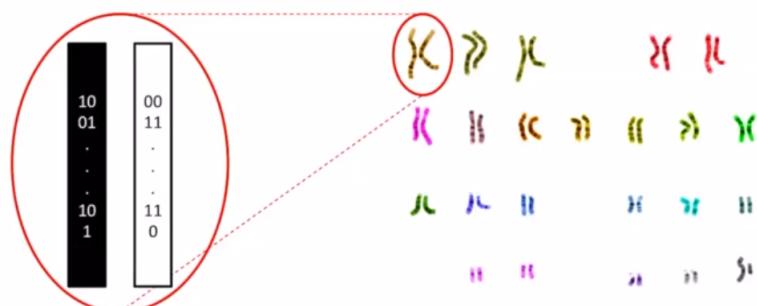
Si hanno $S_1 = acctacga$ e $S_2 = accgacga$

Abbiamo uno SNP nel locus 4. Ipotizzando che il 99.9% della popolazione siano come l'individuo con la sequenza S_1 , abbiamo che la base *t* è l'allele di maggioranza, mentre la base *g* è l'allele di minoranza.

Gli aploidi vengono rappresentati con dei vettori binari dove 0 rappresenta l'allele di maggioranza e 1 l'allele di minoranza, poiché i siti sono biallelici, cioè nella popolazione (*umana, le piante ad esempio sono poliploidi*) si osservano al più due alleli possibili. Dunque un aplotipo è un vettore binario, mentre il genotipo è un vettore di insiemi con due valori binari.

ESEMPIO

Siano gli aploidi 1 1 0 0 1 e 1 0 0 0 1, il genotipo corrispondente è
 $\{1, 1\} \{1, 0\} \{0, 0\} \{0, 0\} \{1, 1\}$



IL DATO DI SEQUENZIAMENTO

Il *dato di sequenziamento* è quel dato che viene prodotto in laboratorio e viene usato sul calcolatore al fine di assemblare un genoma. Quindi i dati di sequenziamento sono sottostringhe del genoma.

IL SEQUENZIAMENTO

Sequenziare, in bioinformatica, significa determinare la sequenza primaria di una molecola biologica, ovvero la sequenza delle basi $\{A, C, G, T\}$ in un DNA oppure la sequenza degli amminoacidi in una proteina.

Il sequenziamento avviene in due passi fondamentali:

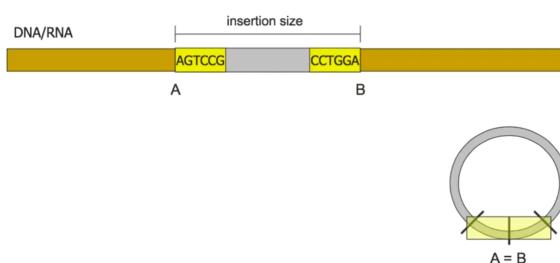
1. **Shotgun sequencing (in laboratorio)**: è l'esperimento in laboratorio che consiste nel prendere la molecola biologica da sequenziare e frammentarla. Questi frammenti non hanno più una localizzazione, non sappiamo da che posizione proviene ogni frammento. I frammenti prendono il nome di **reads**.
2. **Fragment assembly (in silicio)**: in questo step vengono riassemblati i frammenti ottenuti al primo step utilizzando un calcolatore (*vengono assemblate delle stringhe*). I frammenti devono essere ricomposti nella sequenza primaria di origine.

SHOTGUN SEQUENCING

Quello che si ottiene da un esperimento di sequenziamento di una molecola di DNA (o RNA) è un **single - end read** che è semplicemente una sottostringa della molecola di partenza. Generalmente otteniamo tanti single - end read senza conoscere però la loro origine.



Un altro tipo di dato che si ottiene da un esperimento di sequenziamento è una **paired - end / mate - pair reads** che considera una sotto - regione della molecola che prende il nome di **insertion size**. Per ottenere questo dato si prende la insertion size, la si circolarizza (*facendo in modo che i punti A e B si congiungano*) e si sequenzia la zona attorno alla congiunzione A - B, ottenendo quindi un frammento che copre la chiusura attorno ai due punti A e B, ottenendo di fatto la sequenza primaria di un prefisso della insertion size e la sequenza primaria di un suffisso di quest'ultima; abbiamo quindi due frammenti accoppiati che, rispetto alla molecola di partenza, hanno una distanza conosciuta che è l'insertion size. L'informazione derivata dalla insertion size risulta poi molto utile per l'assemblaggio dei frammenti.



Abbiamo un **pair - end** quando l'insertion size è al più mille basi (*insertion size* $\leq 1000 \text{ bp}$) e quando il primo frammento (*quello del prefisso*) è letto in **forward** e il secondo (*quello del suffisso*) è letto in **reverse**.

Quindi nel caso di un pair - end, nell'esempio avremmo i frammenti *AGTCCG* e *AGGTCC*.

Il **mate - pair** ha invece delle insertion size più lunghe (*insertion size* $\leq 5000 \text{ bp}$) e legge i frammenti in modo diverso: il primo in reverse e il secondo in forward. Nell'esempio avremmo i frammenti *GCCTGA* e *CCTGGA*.

TECNOLOGIE DI SEQUENZIAMENTO - METODO SANGER

Lo shotgun sequencing è il procedimento che permette di amplificare una molecola e frammentarla e quindi ottenere i frammenti di molecola da sottoporre individualmente al sequenziamento.

Questo sequenziamento individuale può essere fatto con le seguenti tecnologie:

1. **1977**: Metodo tradizionale Sanger (*prima generazione*)

Questa è una tecnologia singola ed è quindi un metodo unico ed è chiamato anche *Chain - Termination Method*.

Questo è anche il metodo usando nel progetto *Genoma Umano*.

2. **2000**: Next - Generation Sequencing (*NGS*)

3. **2005** Next - Next - Generation Sequencing

Quello che andremmo ad analizzare è proprio il metodo Sanger visto che le tecnologie successive sono molto più complesse. Ideato nel 1977 da *Frederik Sanger* ha portato alla collaborazione tra gruppi di ricerca a livello internazionale e allo sviluppo delle prime **banche di dati genomiche** e ai primi software per processare le sequenze come **BLAST** (*Basic Local Alignment Software Tool*).

Possiamo vedere questo metodo, a livello operativo, come composto da una serie di componenti quali:

- **DNA Template**: è semplicemente il frammento ottenuto per shotgun da sequenziare
- **Primer**: questo metodo richiede di conoscere le prime basi del frammento; il primer rappresenta una piccola molecola di poche basi che rappresenta l'inizio del frammento.
- **DNA Polimerasi**: l'enzima che favorisce la polimerizzazione delle catene di DNA; va a prendere i nucleotidi e gli aggancia seguendo la regola delle basi complementari.
- **Deossinucleosidi**: i nucleotidi. La sigla dei nucleotidi è **dNTP** dove *N* è la base (*A, C, G o T*)
- **Dideoxinucleosidi**: sono i nucleotidi senza l'ossigeno: questi nucleotidi interrompono la polimerizzazione poiché non possono più agganciarsi ad altri nucleotidi, sono quindi presenti in quantità minori. Si indicano con **ddNTP**.

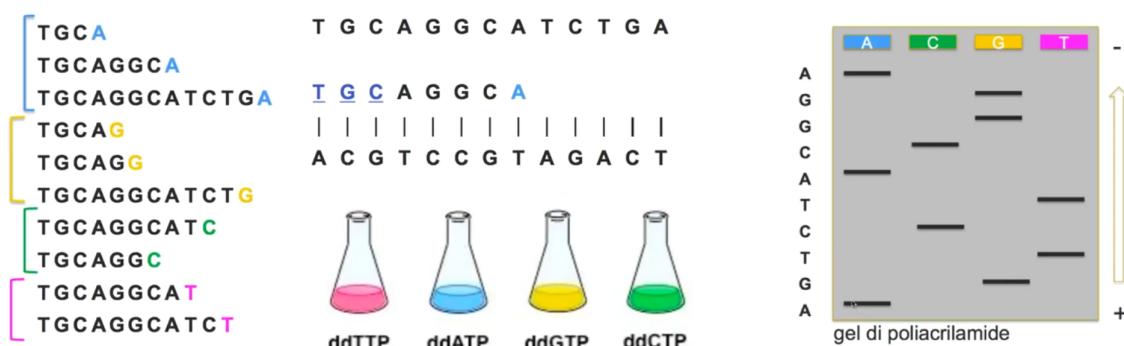
Tutte queste componenti vengono messe in quattro provette, una per base.

PROCEDIMENTO

Queste provette vengono poi inserite in una macchina che esegue varie operazioni:

- **Denaturazione:** la provetta viene portata a 95° e il frammento si divide in due, separando le catene.
- **Annealing:** la temperatura scende a 50° e accade l'annealing del primer, ovvero il primer si attacca alla parte iniziale di una delle due catene.
- **Elongazione:** per opera della DNA Polimerasi che prende i dNTP allungando la catena iniziata dal primer finché casualmente non prende un ddNTP troncando l'elongazione.
- **Terminazione:** quando la DNA Polimerasi prende un ddNTP si ha la fase di terminazione. A questo punto la procedura si ripete, denaturando ancora le catene ottenute e ripetendo il tutto.

Alla fine quello che otterremmo è, per ogni provetta, tutti i frammenti che finiscono con la rispettiva base.



Successivamente le provette vengono inserite in un gel di poliacrilamide che viene sottoposto ad un campo elettrico; in questo modo i frammenti si separano in base alla lunghezza, in particolare i frammenti più lunghi saranno più vicini al polo positivo del campo elettrico. Leggendo poi la lastra di gel dal basso verso l'alto possiamo ricostruire la sequenza di partenza.

Questo era il metodo manuale, al giorno d'oggi si utilizza la **elettroforesi capillare** che permette di fare tutto ciò con un'unica provetta, che tramite un tubo contenente un gel poroso e un laser, riesce a leggere i terminatori nell'ordine giusto producendo un **cromatogramma** che verrà poi letto da un software che ricostruirà la sequenza primaria del DNA Template.

Il metodo Sanger produce un dato con una **qualità** elevata (*ha degli errori sistematici, conosciuti*), una **coverage** bassa di circa 5x / 8x (*la coverage è, data una singola base del genoma, il numero di read che coprono quella specifica base*) e un'ottima **lunghezza** (*fino a 1000 bp*).

Tuttavia è un metodo molto costoso e lento (*3 miliardi di dollari per il Progetto Genoma Umano*).

Una formula per calcolare la coverage è $c = n \cdot l / L$ dove n è il numero di read, l è la lunghezza dei reads e L è la lunghezza della molecola.

TECNOLOGIE NGS

Queste sono tecnologie ideate a partire dai primi anni 2000 e sono altamente parallele e quindi producono dei dati massivi con **qualità** variabile, **coverage** elevatissima e **lunghezza** fino a 300 bp. Queste tecnologie sono veloci e in genere poco costose (*1500\$ per sequenziare un genoma*).

Le principali tecnologie di questo tipo sono **Illumina (Solexa)** e **Ion Torrent - Life Technologies**.

Le tecnologie NNGS producono invece le cosiddette long reads che superano anche quelle di Sanger (*10-15 mila bp*).

CONSEGUENZE IMPORTANTI

Le principali conseguenze importanti delle tecnologie NGS ed NNGS sulla bioinformatica sono

- L'esplosione dei dati visto il parallelismo e il costo ridotto
- Nuovi problemi computazionali al di là dell'assemblaggio come la quantificazione dei trascritti, la predizione di nuovi eventi di splicing, la filogenesi tumorale, l'aplotipizzazione, il variant calling e la rappresentazione di più genomi.

QUALITÀ DEL DATO NGS

Dato che i read NGS sono relativamente corti è fondamentale conoscere la qualità del dato per questo i sequenziatori NGS forniscono la sequenza primaria del read e la sequenza delle qualità delle basi del read associando ad ogni base il **Phred Quality Score** calcolando la probabilità $0 \leq p_e \leq 1$ che la base sia errata. Il Phred Quality Score sarà poi $q = -10 \log_{10} p_e$.

Ai fini pratici una base con $q \geq 50$ viene considerata corretta, che corrisponde ad una $p_e = 0,00001$, ovvero consideriamo corrette le basi con una probabilità di errore di al più 0,001%.

Per questo motivo i read NGS subiscono un pre - processing che va a correggere gli errori, a scartare i read di bassa qualità o fa un trimming dei read (*mantiene solo la parte buona*).

FASTQ

Il formato FASTQ è un formato plain text che associa ad una sequenza la qualità di ogni sua singola base ed è il formato standard di output dei sequenziatori NGS e ha estensione **.fq* o **.fastq*.

Ogni frammento viene memorizzato in quattro record:

1. *header* contenente il suo identificatore ed inizia con @
2. *Sequenza primaria del read*
3. *header* della sequenza dei *phred values*, inizia con +
4. *Sequenza dei phred values*, uno per base convertiti in ASCII

FRAGMENT ASSEMBLY

L'assemblaggio dei frammenti ottenuti per sequenziamento risulta più facile nel caso dell'assemblaggio di un genoma piuttosto che di un trascritto.

Il processo di fragment assembly prende in input un set **R** di *reads* e produce in output una sequenza primaria della molecola originaria (*spesso non è un'unica sequenza*).

Esistono fondamentalmente due tipi di assemblaggio:

1. **ASSEMBLAGGIO DE - NOVO**: si hanno a disposizione solo i frammenti e bisogna assemblarli come fosse un puzzle. Ci concentreremo su questo tipo di assemblaggio.
2. **ASSEMBLAGGIO REFERENCE - BASED (RESEQUENCING)**: oltre ai frammenti abbiamo anche una sequenza di riferimento che non è quella di origine ma è molto simile.

ASSEMBLAGGIO DE NOVO

L'assemblaggio de novo si divide in due step:

1. *Determinazione dei contigs*: questo primo step consiste nell'ottenere dei *pezzi* continui i più lunghi possibili; non otterremmo la sequenza intera, ma per pezzi.
Questo è un compito difficile per via degli **errori di sequenziamento** e per via delle **ripetizioni sul genoma** (*che dipendono dal genoma, non dalle tecnologie di sequenziamento*).
2. *Scaffolding*: consiste nel prendere i *contigs* e assemblarli in modo da ricoprire i buchi e ottenere idealmente la sequenza primaria.

DETERMINAZIONE DEI CONTINGS

Il passo principale sul quale ci concentreremo è la determinazione dei contings e per farlo esistono due approcci principali:

1. **Overlap Layout Consesus (OLC)**: Si basa sul concetto di *overlap*; prende le reads e cerca di individuare quali sono le sovrapposizioni. Le coppie di read che sono in sovrapposizione vengono assemblate. Per individuare gli overlap viene usato un grafo.
2. **Grafo di de Bruijn (dBG)**: Si basa sui *k*-meri, dove un *k*-mero è una sottostringa di lunghezza *k*. Scomponi i frammenti in *k*-meri; per assemblare le read, le scomponi ed assembila i *k*-meri. Potrebbe sembrare inefficiente mentre invece è più efficiente del primo.

SHORTEST SUPERSTRING PROBLEM

Il problema teorico che sta alla base dell'assemblaggio è quello della Shortest Superstring Problem definito come segue sotto ipotesi dell'assenza di errori di sequenziamento (*che non è troppo rigida, spesso si correggono gli errori prima dell'assemblaggio*):

INPUT: r_1, \dots, r_n stringhe su $\Sigma = \{A, C, G, T\}$

OUTPUT: stringa G di minima lunghezza tale che ogni stringa r_i sia sottostringa di G

La stringa G rappresenta il genoma, ovvero la stringa che contiene tutti i frammenti. Si richiede che sia di minima di lunghezza poiché altrimenti avremmo come soluzione la concatenazione di tutti i frammenti.

Esempio

Prendiamo $\Sigma = \{0, 1\}$ e le stringhe 000, 001, 010, 100, 101, 110, 111

Due possibili soluzioni sono 1000101110 e 0001110100.

Il problema non ammette un'unica soluzione.

OVERLAP LAYOUT CONSENSUS

Questo approccio prevede tre step:

1. **Calcolo degli overlap tra i reads:** si va a scoprire i vari overlap tra tutte le coppie di read.
2. **Costruzione dell'Overlap Graph (OG) (Layout):** dispone i frammenti lungo il candidato genoma.
3. **Visita del grafo (Consesus):** l'assemblaggio dei read viene ottenuto visitando il grafo. Questo passo implicherà la ricerca di un *cammino hamiltoniano*.

Questo approccio è stato usato per ricostruire il primo genoma umano. Questo approccio funziona bene se i read sono pochi e lunghi (*come quelli prodotti dal metodo Sanger*).

Noi ci concentreremo maggiormente sui punti 2 e 3, non trattiamo i metodi del calcolo degli overlap.

OVERLAP: date due read r_1 ed r_2 , chiamiamo overlap tra queste due read il più lungo **suffisso** di r_1 che si ripete in un **prefisso** di r_2 (*o viceversa*).

Notiamo nell'esempio che il primo *non* è un overlap poiché non è il suffisso più lungo che si ripete come prefisso nell'altro read.

ATGGAGAGAGA	ATGGAGAGAGA
AGAGAGAAGT	AGAGAGAAGT

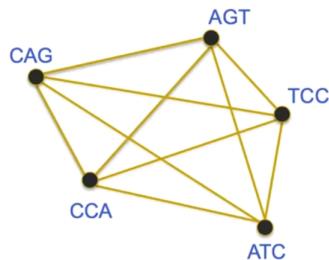
Ci interessa il suffisso/prefisso più lungo poiché poi quando andiamo a costruire la super stringa minima possiamo usare questo *aggancio* più profondo per la costruzione di quest'ultima.

EXTENTION: la parte del read che abbiamo più a destra che *esce* rispetto all'overlap. Nel caso dell'esempio precedente l'extention è AGT.

ASSEMBLY: l'assembly dei read r_1 ed r_2 è dato dal read a sinistra a cui viene concatenata l'estensione del loro overlap. Nell'esempio precedente l'assembly è ATGGAGAGAGAAGT. L'assembly è la minima super stringa dei due read. Questa operazione viene poi eseguita ricorsivamente per tutti i read.

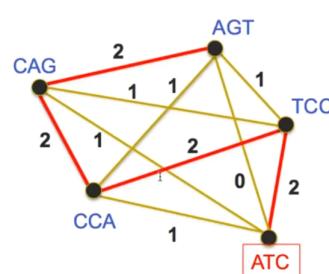
Definiamo ora il **grafo completo dei reads** (*non orientato*) come

$$G = (V, E) \text{ dove } V = \{r_1, \dots, r_n\} \quad E = \{(r_i, r_j) \mid r_i, r_j \in V\}$$



Ricordiamo che un **cammino hamiltoniano** è un cammino che tocca tutti i vertici una e una sola volta.

Un possibile cammino hamiltoniano è $CH = < TCC, AGT, CCA, ATC, CAG >$; se andiamo a concatenare i nodi del cammino otteniamo una super stringa (*non è detto che sia la minima però*).



Se ora andiamo a modificare il grafo aggiungendo su ogni lato un peso corrispondente alla lunghezza dell'overlap tra i due read che stanno sui vertici e cerchiamo il cammino hamiltoniano di peso massimo, otterremmo una super stringa che tiene conto dei vari overlap e quindi otterremmo la **super stringa minima**.

In questo caso il cammino hamiltoniano di peso massimo è $CH = < ATC, TCC, CCA, CAG, AGT >$ e la super stringa minima sarà $ATCCAGT$ ottenuta concatenando i nodi del cammino sovrapponendo gli overlap.

Chiaramente, basandosi questo problema sulla ricerca di un cammino hamiltoniano, l'overlap layout consensu è un problema NP - Completo. Per la ricostruzione del genoma umano sono state utilizzate delle euristiche.

Quello che abbiamo visto ora tuttavia è un'estensione del grafo di overlap e per introdurre quest'ultima dobbiamo modificare la definizione di overlap data prima introducendo un orientamento: diremo che l'**overlap** $ov(r_1, r_2)$ è il più lungo suffisso di r_1 che si ripete in un prefisso di r_2 . Indicheremo l'estensione con $ex(r_1, r_2)$.

L'Overlap Graph (*orientato*) è quindi

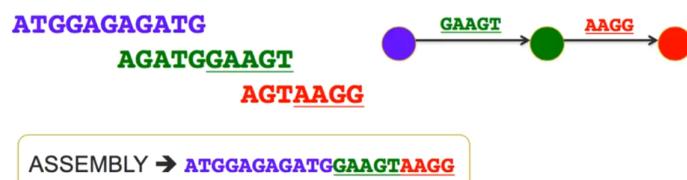
$$OG = (V, A) \text{ con } V = \{r_1, \dots, r_n\} \text{ e } A = \{(r_i, r_j) \mid r_i, r_j \in V \wedge ov(r_i, r_j) \neq \epsilon\}$$

Sono quindi collegare solo le read che hanno un overlap non nullo (*a volte si richiede anche una soglia minima di lunghezza*). Inoltre, l'arco (r_i, r_j) è etichettato da $ex(r_i, r_j)$.

Di conseguenza visitando il grafo si costruisce la super stringa minima prendendo l'etichetta del primo nodo e concatenando le etichette degli archi visitati

$$\text{Assembly} = r_1 \cdot ex(r_1, r_2) \cdot \dots \cdot ex(r_{n-1}, r_n).$$

Chiaramente possono esistere degli archi transitivi nell'overlap graph (*ad esempio se la stringa viola contenesse anche un prefisso di quella rossa come suffisso avremmo un arco dal nodo viola a quello rosso*) che però non cambiano il risultato della visita e quindi vengono rimossi prima della visita poiché potrebbe precludere percorsi alternativi. Il grafo ottenuto rimuovendo gli archi riducibili si chiama **String Graph**.



GRAFO DI DE BRUIJN

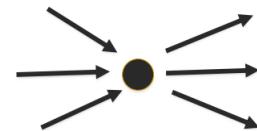
Anche questo approccio prevede tre step:

1. **Estrazione dei k -meri dai reads:** estrae tutte le sottostringhe di lunghezza k .
2. **Costruzione del grafo di de Bruijn:** costruisce il grafo coi k -meri.
3. **Visita del grafo:** anche qui l'assemblaggio viene ottenuto visitando il grafo. Implica la ricerca di un *cammino euleriano*.

Questo approccio funziona molto bene con frammenti NGS.

Definiamo innanzitutto il **cammino euleriano** su un grafo $G = (V, E)$ come il cammino che percorre tutti gli **archi** del grafo una e una sola volta. Questo è un problema più semplice del *cammino hamiltoniano* e ha soluzione polinomiale.

TEOREMA: Un grafo G ammette un *cammino euleriano* se e solo se è bilanciato, cioè se per ogni $v \in V$ $\text{in}(v) = \text{out}(v)$, ovvero se ogni vertice ha un numero di archi entranti pari al numero di archi uscenti.



PROOF

1. Se esiste un ciclo euleriano, il grafo è bilanciato

Difatti, se per assurdo un nodo non fosse bilanciato significherebbe che uno degli archi entranti o uscenti da esso non potrebbe essere percorso senza percorrere più di una volta uno degli altri archi.

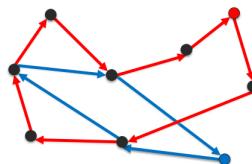
2. Se il grafo è bilanciato, allora esiste un cammino euleriano

Per la dimostrazione vediamo l'algoritmo di costruzione del cammino euleriano:

1. Scomposizione in cicli

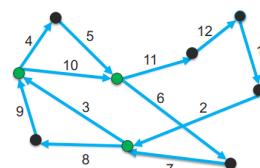
Parte da un arco e percorre archi successivi fino a trovare un ciclo:

1. Ha percorso tutti gli archi quindi si ferma
2. Esistono archi non ancora percorsi, quindi parte da un arco non ancora percorso e cerca di trovare un nuovo ciclo



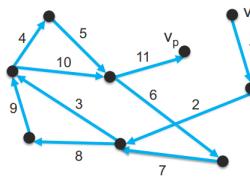
2. Composizione dei cicli trovati

Parte da un arco (*di un determinato ciclo*) e percorre gli archi del grafo uscendo se possibile su un ciclo diverso.



L'algoritmo ha un tempo $O(|V| + |E|)$

Può esistere un cammino euleriano anche per i grafi **semi - bilanciati**, ovvero i grafi tali per cui esiste un solo nodo v_s per cui $out(v_s) = in(v_s) + 1$ chiamato *nodo sorgente* ed esiste un solo nodo v_p per cui $in(v_p) = out(v_p) + 1$ chiamato *nodo pozzo*.



L'approccio con il grafo di de Bruijn ha come obiettivo proprio quello di assemblare una sequenza di DNA attraverso la determinazione del cammino euleriano in un grafo semi - bilanciato.

DEFINIZIONE

Come abbiamo detto prima, un k -mero di un stringa r su alfabeto Σ è un elemento di Σ^k che occorre in r , ovvero una sottostringa di r di lunghezza k .

- GTGC** Come nel caso dell'OLC, l'assemblaggio di due k -meri m_1 ed m_2 che hanno overlap lungo $k - 1$ è il $(k + 1)$ -mero dato dalla concatenazione tra m_1 e l'estensione di m_2 rispetto all'overlap.
- TGCA**
- GTGCA**

DEFINIZIONE

Definiamo anche lo **spettro di ordine k** di una stringa come il *multiset* di tutti i suoi k -meri.

Lo **spettro di ordine k** di una *collezione* di stringhe invece è il *multiset* di tutti i k -meri delle stringhe della collezione (*doppioni inclusi*).

ACGTGCCGTG

Spettro di ordine 4:

{ACGT, CGTG, GTGC, TGCC, GCCG, CCGT, CGTG}

ACGTGCCGTG, CCGTCGGTAA

Spettro di ordine 4:

{ACGT, CGTG, GTGC, TGCC, GCCG, CCGT, CGTG, CCGT, CGTC, GTGC, TCGG, CGGT, GGTA, GTAA}

Possiamo ora definire il **Grafo di de Bruijn di ordine k** ; esistono due definizioni (*equivalenti*)

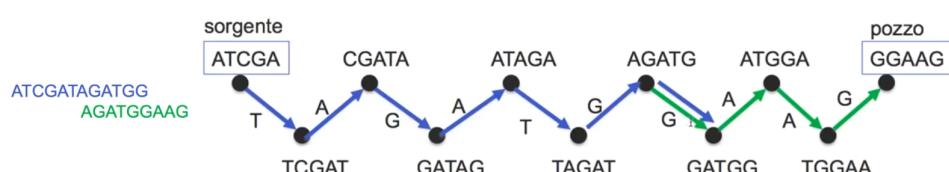
1. Definizione nodo - centrica

In questa definizione un Grafo di de Bruijn di una collezione di reads $\{r_1, \dots, r_n\}$ è il grafo $G = (V, A)$ dove

- V è l'**insieme** degli elementi dello spettro di ordine k della collezione di reads.
- ATTENZIONE: è un insieme; a differenza della collezione non presenta i duplicati.
- Esiste un arco dal nodo v_1 al nodo v_2 se e solo se
 1. Il suffisso di v_1 lungo $k - 1$ occorre come prefisso di lunghezza $k - 1$ su v_2 (quindi se v_1 e v_2 hanno un overlap di lunghezza $k - 1$).
 2. Il $(k + 1)$ -mero ottenuto assemblando v_1 e v_2 è sottostringa in almeno un read della collezione.

In particolare, l'arco viene etichettato con $ex(v_1, v_2)$

NOTA: questo è un multi grafo (*semi - bilanciato*), può esserci un arco duplicato tra due nodi che però vale come un unico arco (*read con overlap di lunghezza maggiore di k*).

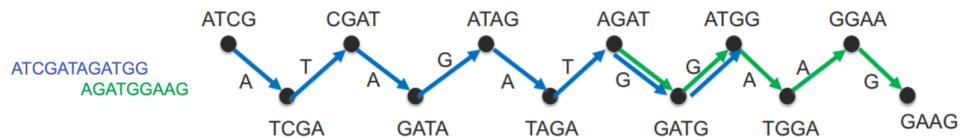


2. Definizione arco - centrica

In questa definizione un Grafo di de Bruijn di una collezione di reads $\{r_1, \dots, r_n\}$ è il grafo $G = (V, A)$ dove

- V è l'**insieme** degli elementi dello spettro di ordine $k - 1$ della collezione di reads.
 - A rappresenta l'**insieme** degli elementi nello stesso spettro di ordine k
- Cioè l'arco dal nodo v_1 al nodo v_2 è tale per cui:
1. Il suffisso di v_1 lungo $k - 2$ è uguale al prefisso di v_2 lungo $k - 2$ (*overlap*)
 2. Il k -mero ottenuto assemblando v_1 e v_2 è sottostringa in almeno un read della collezione.

L'arco inoltre è etichettato da $ex(v_1, v_2)$.



NOTA: Il de Bruijn graph *arco - centrico* di ordine k è uguale al de Bruijn graph *nodo - centrico* di ordine $k - 1$

PRESTARE ATTENZIONE A

- *Loop*

Prendiamo per esempio la stringa AAAAAAAA e $k = 4$; notiamo che ha 6 k -meri tutti uguali, quindi il suo grafo risulterà in unico nodo con un cappio. Il problema è che rischiamo di far collassare la stringa in un unico k -mero se il k del grafo è troppo piccolo.

- *Ripetizioni*

Se scegliamo un k minore della lunghezza delle stringhe che si ripetono nel genoma potremmo avere degli archi che renderebbero il grafo non bilanciato.

Altri problemi dati dalle ripetizioni potrebbero essere anche la presenza di diversi cammini euleriani nel grafo e quindi di diversi assemblaggi.

Solitamente per l'assemblaggio del genoma viene scelto un k pari a 32.

ATTENZIONE: un k troppo alto porterebbe invece alla sconnessione di alcuni nodi, creando quindi un grafo sconnesso.



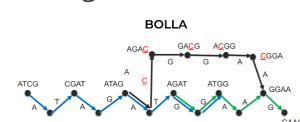
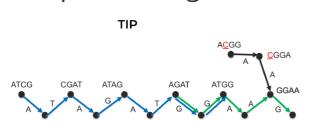
- *Errori di sequenziamento*

 - *Tips*

Una *tip* è un percorso lineare che si introduce (o esce) poi nel grafo. Prima dell'assemblaggio si scartano le *tips*.

 - *Bubbles*

Una bolla è una *tip* che esce e poi rientra nel grafo: generalmente si sceglie il percorso con più coverage. Le bolle possono essere dovute anche agli SNP.



CONFRONTO

Confrontiamo l'overlap graph con il de Bruijn graph;

- Il primo è più lento chiaramente per via del cammino hamiltoniano che non ha soluzione polinomiale come il cammino euleriano.
- L'overlap graph tratta bene i read lunghi (*Sanger*), mentre il de Bruijn graph tratta bene le read corte.
- L'overlap graph tuttavia rispetto al de Bruijn graph (*almeno per k bassi*) è poco sensibile alle ripetizioni.
- L'overlap dell'overlap graph è variabile rispetto all'overlap dei k -meri e per ciò è più adattativo.
- A volte con i de Bruijn graph si può perdere la coerenza con i reads.
- Il tempo di costruzione del de Bruijn graph è $O(n)$ mentre lo spazio è $O(\min(N, G))$
Mentre il tempo dell'overlap graph è $O(N + a)$ e lo spazio è $O(N + a)$
 N = lunghezza totale dei read G = lunghezza genoma da ricostruire
 a = numero di overlaps $O(n^2)$ con n numero di reads.

SEQUENZIAMENTO PER IBRIDAZIONE

Un metodo alternativo proposto nel 1988 per il sequenziamento del genoma è quello basato sulla ibridazione che è la reazione che si genera tra due catene complementari di DNA.

Il principio si basa su una superficie matriciale su cui vengono attaccati dei *probes*, ovvero delle sequenze corte di lunghezza nota; viene poi applicata una soluzione contenente il DNA template da sequenziare che è stato etichettato con una sostanza fluorescente.

Ogni *probe* ibridizza con la sua versione complementare del frammento e tramite esame radiografico si determinano i *probes* che hanno ibridizzato col frammento per ricostruire lo spettro complementare del frammento da cui si può poi ricostruire il frammento tramite il de Bruijn graph.

STRUTTURE DATI IN BIOINFORMATICA

La gestione dei k -meri è fondamentale in bioinformatica vista la grandi quantità di dati prodotta in questo ambito.

Uno dei concetti chiavi per trattare grandi moli di dati sono i **fully-text-index** che, dato un testo $T = t_1 \ t_2 \ \dots \ t_n$, è una struttura dati costruita per tenere il testo in memoria e su cui si possono fare delle query che cercano delle posizioni e delle frequenze di sottostringhe in T . La particolarità di queste strutture è che hanno tempo sub-lineare rispetto a n (*si pensi alla BWT*).

HASHING

Una struttura dati tipica per gestire questi dati sono le tabelle di hash.

L'idea principale è che abbiamo delle chiavi k_i che vengono mappate negli ingressi di una tabella tramite una funzione di hash. Più chiavi hashate possono essere mappate sulla stessa cella della tabella, specie se ho tante chiavi (*collisione*).

Solitamente le collisioni sono aspetti negativi delle funzioni di hash, tuttavia in questo caso possono essere positive poiché si possono trovare delle similarità tra gli oggetti che collidono nella stessa posizione.

Questa tecnica prende il nome di **Local Sensitive Hashing**.

L'idea è assegnare un *bucket* ad ogni oggetto (*tempo costante*) tramite la funzione di hash. Questo risulta utile per fare clustering di genomi simili.

Questo viene fatto in due step:

1. Le sequenze (*o parti di esse*) vengono riassunte in **sketch**, ovvero in rappresentazioni più piccole.
2. Si confrontano gli sketch come chiavi di tabelle di hash, in modo da trovare le sequenze simili. Solitamente si ussa la *Jaccard distance* per il confronto.

Funzioni di hash che assegnano lo stesso hash a elementi simili si dicono **locality sensitive**, in particolare si ha $P(h(x) = h(y)) = s(x, y)$. Questo si presta molto bene alla distanza di Jaccard calcolata come $(A \cap B)/(A \cup B)$.

Per confrontare due sequenze x e y si usano t funzioni di hash andando a costruire sketch di $x = < h_1(x) \ \dots \ h_t(x) >$ e di $y = < h_1(y) \ \dots \ h_t(y) >$. A questo punto la somiglianza tra x e y è $s(x, y) = \#h_i(x) = h_i(y)/t$. Praticamente si conta quante volte le due sequenze collidono.

Questo viene fatto per le sequenze in bioinformatica estraendo i k -meri dalle sequenze (*che corrispondono a degli oggetti in un insieme*) e si ordinano lessicograficamente, si fanno delle permutazioni e si prende il valore minimo, che corrisponderà ad un certo k -mero.

STRUTTURE DI INDICIZZAZIONE

Le strutture di indicizzazione sono fondamentali in bioinformatica essendo i genomi molto lunghi e per via delle numerose read.

Vedremo Suffix Tree, il Suffix Array (*introdotta come supporto del suffix tree che occupa molto spazio*), la Longest Common Prefix Array e la Burrow - Wheeler Transformation.

NOTAZIONI

Useremo Σ per indicare un alfabeto finito, T per indicare un testo di n caratteri su Σ .
 $T[j : q]$ è la sottostringa di T dalla posizione j alla posizione q e $T[j :]$ è il suffisso di T che inizia in posizione j .

SUFFIX ARRAY

Il Suffix Array di un testo T \$-terminato di n caratteri è un array S di n interi tale che $S[i] = j$ se e solo se $T[j :]$ è l' i -esimo suffisso nell'ordinamento lessicografico dei suffissi di T .

Praticamente ci dice la posizione di inizio dell' i -esimo suffisso. Permette di fare una ricerca binaria per trovare un pattern in un testo in $O(m \log n)$.

Si può accelerare salvando ad ogni iterazione la lunghezza del più lungo prefisso tra P e $T[S[L] :]$ e P e $T[S[R] :]$ in L e R : quando partirà nuovamente il confronto con il suffisso di mezzo si potranno saltare i primi $\min(L, R)$ caratteri per via dell'ordinamento lessicografico. Questo accelerante non cambia l'ordine di grandezza dell'algoritmo però.

Esiste anche il *suffix array inverso* S^{-1} tale che $S^{-1}[j] = i$ sse $T[j :]$ è l' i -esimo suffisso. Ci dice in che posizione si trova lessicograficamente il j -esimo suffisso.

SUFFIX TREE

Il Suffix Tree consente di risolvere in tempo lineare molti problemi su stringhe ma occupa molta memoria in quanto contiene informazione *ripetuta* del testo.

Il Suffix Tree di un testo T lungo n è un albero radicato dove le foglie (n) sono etichettate dagli interi da 1 a n e ogni nodo interno ha almeno due figli. Gli archi sono etichettati da una sottostringa di T e non esistono due archi uscenti dallo stesso nodo che hanno etichetta che inizia con lo stesso carattere.

La concatenazione delle etichette lungo il percorso dalla radice alla foglia j è uguale al suffisso $T[j :]$.

NOTA: Il Suffix Tree di un testo esiste se e solo se nessun suffisso di T occorre come prefisso di qualche altro suffisso. Non potremmo più non avere archi uscenti dallo stesso nodo che iniziano con lo stesso carattere.

Tuttavia possiamo imporre la condizione che dice che il Suffix Tree esiste se e solo se l'ultimo carattere di T non occorre in nessun'altra parte di T . Da qui nascono i testi \$-terminati.

Dato un nodo del Suffix Tree w , la sua path label L_w è la concatenazione delle etichette lungo il percorso dalla radice a w .

La Path Label della foglia j è il suffisso $T[j :]$, mentre la Path Label di un nodo interno w sarà prefisso in almeno un suffisso di T .

Chiaramente due nodi diversi non possono avere la stessa path label.

Dato un nodo w , la String Depth è la lunghezza della sua Path Label L_w .

Un **dummy node** è un falso nodo che separa l'etichetta di un arco in due stringhe.

Per un nodo interno w , le k foglie del sottoalbero radicato in w danno le posizioni di inizio dei k suffissi che condividono path label L_w come prefisso.

Si può cercare un pattern P lungo m in un testo lungo n in cui occorre k volte in tempo $O(m + k)$ cercando il nodo (*anche dummy*) w che ha P come path label ($O(m)$) e si visita il suo sottoalbero elencandone le foglie ($O(k)$).

Per costruire il Suffix Array dal Suffix Tree ci basta eseguire una visita in profondità dell'albero scegliendo però i nodi da esplorare in ordine lessicografico.

LONGEST COMMON PREFIX ARRAY

Dato un testo T di n caratteri, il LCP Array è un array di n interi tale che $LCP[i] = l$ se e solo se l è la lunghezza del più lungo prefisso comune tra $T[S[i] :]$ e $T[S[i - 1] :]$.

Esiste anche la ***lcp function***: $lcp(i, j)$ restituisce la lunghezza del più lungo prefisso comune tra $T[S[i] :]$ e $T[S[j] :]$ (*assumiamo $i < j$*).

In particolare questo valore sarà il minimo tra $LCP[i+1], LCP[i+2], \dots, LCP[j]$.

Si dice quindi che la $lcp(i, j)$ è una *range minimum query* RMQ(LCP, $i+1, j$).

Unendo suffix array e LCP array è possibile ridurre la complessità dell'algoritmo di ricerca di un pattern in un testo a $O(m + \log n)$.

Ad ogni step della ricerca binaria vengono mantenuti quattro valori:

1. Lunghezza l del più lungo prefisso condiviso tra P e $T[S[L] :]$
2. Lunghezza r del più lungo prefisso condiviso tra P e $T[S[R] :]$
3. Lunghezza del più lungo prefisso condiviso tra $T[S[L] :]$ e $T[S[q] :]$, ovvero $lcp(L, q)$
4. Lunghezza del più lungo prefisso condiviso tra $T[S[q] :]$ e $T[S[R] :]$, ovvero $lcp(q, R)$

Mentre i primi due sono facilmente calcolabili, gli ultimi due devono essere ricavati da un preprocessing del testo in modo da potervi accedere in tempo costante.

Abbiamo tre casi:

$$1. l = r$$

Visto che le due lunghezze sono uguali, P condivide con $T[S[q] :]$ sempre un prefisso lungo $l = r$.

P viene confrontato con $T[S[q] :]$ partendo quindi dalla posizione $l + 1$.

$$2. l > r$$

In questo caso P condivide con $T[S[q] :]$ un prefisso lungo r .

Dobbiamo controllare:

$$1. lcp(L, q) > l$$

Questo significa che $P[l + 1] \neq T[S[L] :][l + 1]$ e che
 $T[S[L] :][l + 1] = T[S[q] :][l + 1]$

Di conseguenza, non dobbiamo nemmeno confrontare P e $T[S[q] :]$.

Il nuovo intervallo di ricerca deve essere q, R .

l e r rimangono inalterati.

$$2. lcp(L, q) < l$$

Questo significa che $P[lcp(L, q) + 1] = T[S[L] :][lcp(L, q) + 1]$ e
 $T[S[L] :][lcp(L, q) + 1] \neq T[S[q] :][lcp(L, q) + 1]$

Anche qui non devo confrontare il pattern col suffisso di mezzo.

In questo caso il nuovo intervallo di ricerca è L, q .

l rimane inalterato, r va ricalcolato e sarà $\text{lcp}(L, q)$.

3. $\text{lcp}(L, q) = l$

Caso più sfortunato, non si può dire niente.

Si confronta P con $T[S[q] :]$ dalla posizione $l + 1$ e si decide il nuovo intervallo di ricerca.

3. $r > l$, duale del caso due.

Il preprocessing avviene in tempo $O(n)$ e calcola $\text{lcp}(L, p)$ e $\text{lcp}(p, R)$ per ogni intervallo $[L, R]$ possibile.

Questo viene fatto costruendo il *Search Interval Tree* che ha come nodi tutti i possibili intervalli di ricerca $[L, R]$. La radice è l'intervallo iniziale $[1, n]$ e ogni nodo interno ha un *left child* $[L, p]$ e un *right child* $[p, R]$.

Si hanno quindi n foglie $[f, f + 1]$ (*per i testi pari ci sarà anche la foglia [1, 1]*).

Si fa poi una visita in profondità di questo albero e, risalendo, si calcolano i vari $\text{lcp}(L, p)$ e $\text{lcp}(p, R)$.

Per come è fatta la LCP, una volta calcolate le foglie, possiamo calcolare i nodi interni prendendo i minimi delle loro foglie.